22 PRINTER 35 PROG

Free Inside
YOUR ROBOT
Includes Solid state TV camera project

THE BE MPUTER PROJECTS MAGAZINE APRIL 1984

ECTRONICS & COMPUTING

MONTHLY

AN EMAP PUBLICATION

11 REC 15 ERRA-CROSSASSEM. 16 . 22 PRINTER EP44(600) REVIEW
24 ORIC A/D CONVERTER 33 MICRO GRAPHICS 38/1 RCA 1 BOOKLET

USA $2.95
Germany D5.80 85p

DON'T MISS THAT DATE!

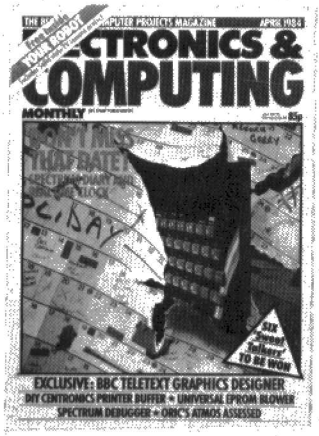
SPECTRUM DIARY AND REAL TIME CLOCK

SIX 'Sweet Talkers' TO BE WON

EXCLUSIVE: BBC TELETEXT GRAPHICS DESIGNER

DIY CENTRONICS PRINTER BUFFER ★ UNIVERSAL EPROM BLOWER

SPECTRUM DEBUGGER ★ ORIC'S ATMOS ASSESSED


THE BEST COMPUTER PROJECTS MAGAZINE
APRIL 1984
Free inside YOUR ROBOT
ELECTRONICS & COMPUTING
MONTHLY
EXCLUSIVE: BBC TELETEXT GRAPHICS DESIGNER
DIY CENTRONICS PRINTER BUFFER • UNIVERSAL EPROM BLOWER
SPECTRUM DEBUGGER • ORIC'S ATMOS ASSESSED

COMPUTING contents

VOLUME 4 ISSUE 4

Electronics & Computing Monthly
Scriptor Court, 155 Farringdon Road,
London, EC1R 3AD

Editorial 01-833-0846
Editor Gary Evans
Assistant Editor William Owen
Administration Liz Gregory

Advertising 01-833-0531
Manager Claire Fullerton
Executive Richard Jansz
Classified Andrew King

Production 01-833-0531
Art Editor Jeremy Webb
Make-up Time Graphics
Publisher Alfred Rolington

Distribution
EMAP National Publications

Published by
EMAP Business and
Computer Publications

Printed by
Riverside Press, England

Subscriptions
Electronics & Computing Monthly,
(Subscription Department),
Competition House, Farndon Road,
Market Harborough, Leicestershire.

Electronics & Computing Monthly is normally published on the 13th day of each month

Subscription rates: UK £10.70 incl. post. For overseas rates apply to Subscription Dept., Competition House, Farndon Road, Market Harborough, Leicestershire. Back issues available from: EMAP National Publications (*E&CM* Back Numbers), Bretton Court, Peterborough, PE3 8DZ. Phone: 0733 264666.

ABC

FEATURES

PROJECTS

REVIEW

PLUS

The second part of our multiprocessing feature and the fourth instalment of the SAT 16 project have been held over due to lack of space.

Micromania sweeps nation

A staggering 11 per cent of British households now have at least one computer, and home ownership has more than doubled in the last year according to a survey published in February by Gowling Marketing Services.

In January 1983 microcomputers could be found in 'only' 4.9 per cent of British households, and in the following twelve months all industry projections were exceeded as micromania swept the land, particularly in the run up to Christmas.

It will come as no surprise that Sinclair and Commodore dominate the hardware market; the Gowling report confirms that a small number of very successful computers have elbowed out all competition from late arrivals and early failures, and now six machines account for 87 per cent of all home micros in use.

Reports indicate that half a million Spectrums were sold in the last three months of 1983, and that Sinclair removed advertising from computer publications to try and stem the overwhelming demand.

Demand is likely to be maintained at the same level during 1984, says the report. The signs are that the hardware market will reach maturity by the end of the year and growth will slow down. By that time we can expect a 15-20 per cent level of ownership – that means about 5 million machines.

Needless to say this is good news for the software and peripherals manufacturers. 1984, say Gowling, is likely to be the best year ever for software companies, with arcade and adventure games accounting for an estimated 70 per cent of sales.

Husband first FORTH on BBC micro

David Husband, the man who put FORTH onto the ZX81, has released a 16K multi-tasking FORTH 83 ROM for the BBC micro ahead of the expected Spectrum version.

The decision to bring forward the BBC release followed the completion of the latest FORTH 83 standard by the FORTH standards team. This is the first multi-tasking BBC FORTH system and is marketed by Husband under the new name of Skywave Software.

BBC Multi-FORTH 83 is a 16K 27128 type EPROM, which resides in the sideways ROM area of the BBC. The user can place the ROM in a position of higher priority than the resident BASIC and turn the BBC into a FORTH based machine. Multi-tasking enables the user to execute several programs simultaneously and independently of each other.

The BBC FORTH is compatible with the MOS, enabling the user to enter *CAT to see the disc directory and use all other DFS commands. Block files of over 32K can be stored alongside non-FORTH files on the disc. The system is vectored to enable the user to reconfigure the system and create closed applications.

Multi-FORTH 83 is priced at £40 plus p&p, for which the user receives a standard 6502 assembler, screen editor, stack display (to monitor stack contents for learning and debugging), a number of special interfacing commands, filing and task commands' plus 170 pages of documentation.

Skywave Software is at 73 Curzon Road, Boscombe, Bournemouth BH1 4PW.

Don't hold your breath

Last month our paper review of the Sinclair QL promised a full benchtest for this issue, a promise, that a glance at our contents page will reveal, we have failed to keep. The reason for this is unlikely to surprise anyone, it being that to date we have not managed to get our hands on a machine for evaluation.

While not wishing to dwell on the subject of the QL for too long, Sinclair has already had quite enough publicity for the machine, the fact that even the press have not yet had a chance to see the QL adds fuel to the rumours that Sinclair are having more than a few problems with the computer. While delays in getting products onto the market in any quantity are not new to Sinclair, in the case of both the Spectrum and microdrives, review samples were made available shortly after the official launch of the products.

Sinclair are at present indicating to those who were amongst the first to order a QL that the latest despatch date will be in late April early May. All this makes nonsense of the 28 day delivery claim made in the first batch of adverts for the QL something that the ASA are at present investigating. As was pointed out last month however, the ASA can bark but has little in the way of legal teeth to do more than that.

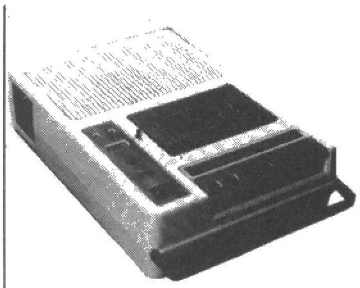
As soon as we can beg, steal or borrow a QL we'll bring you that review but, in the mean time, don't hold your breath.

When comparing the latest home computers to their forerunners, some of the most obvious improvements in design have been made in the areas of the sound and graphics facilities offered. Compare, for example, the Elan with its stereo sound generator and multi-coloured sprite based graphics chips, to the silent and black and white TRS 80 of a few years ago. The reason for the dramatic improvements to the sound and vision sections of home computers is that according to market research, most machines are used to play games. When used in this application it is not difficult to see that the better the sound and graphics, the better the game.

The next stage in the development of a computer's video circuitry will allow the production of graphics that are far more sophisticated than even the best of today's displays. The development will centre around the design of interfaces that will allow computer generated graphics to be superimposed onto the video output of video disc players. Such systems have already made an appearance in the latest video arcade games and are starting to appear on the home computer market in the USA.

The added realism that these techniques bring to simulation games is amazing and when it comes to computer aided education the advantages are also enormous.

GARY EVANS



Thick skinned data recorder

Bell and Howell's new data recorder was designed to take a knocking in schools and colleges, but is now being marketed through consumer outlets to the mass market as a heavy duty peripheral.

All the audio facilities of the educational recorder have been retained. The 3179CX has an internal electret microphone and internal loudspeaker. Two headphone sockets are provided, as well as a socket for remote control of the drive motor by the computer.

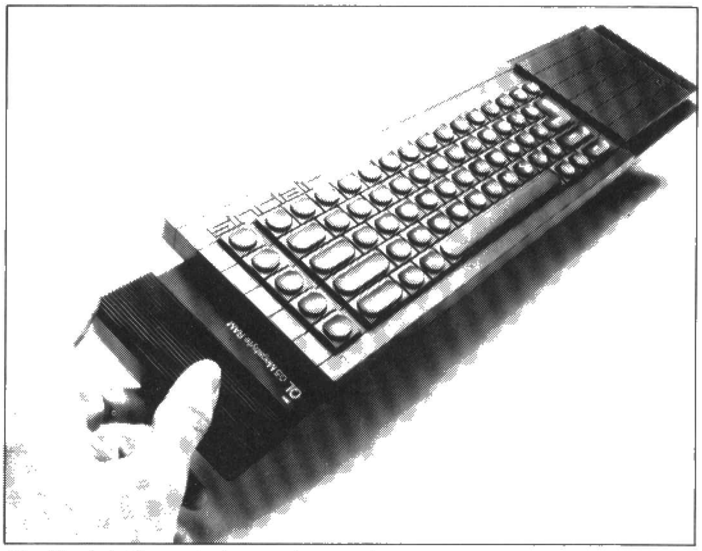
Cue and review modes enable fast forward and rewinding with the play key depressed so that playback begins immediately the wanted section of the tape has been found. Load and save connections to the BBC micro are through an IEC/DIN input/output socket and a 2.5mm jack socket for computer control. For computers requiring a higher output from the recorder when loading (eg the Spectrum) the signal is taken from one of the ¼" headphone sockets, from which up to 4V is available.

Jupiter Ace jacks down the price

Rights to retail the Jupiter Ace, the FORTH based micro which suffered an early demise in late '83, have fallen to Boldfield Computing of Cambridge who have slashed the price of the machine. The good news for Ace owners is that Boldfield will continue to develop software and act as a selling agent to companies producing interfaces and add-ons.

Even better news is that the price has fallen to a ridiculous level – £26 for the Ace with power supply, manual, demonstration cassette and leads. The 16K RAM pack costs a further £20 and software £3 per cassette. A lightning calculation reveals that for £44 it is now possible to buy an excellent control system which three months ago would have set you back a mean £125.

28 days – quaint linguistics!



The Sinclair QL: an industry of paper tigers has grown up to support the elusive machine.

Sinclair Research has been forced to abandon its original claim of 28 days delivery for the QL, and prospective customers should have received a letter signed by managing director Nigel Searle which promises delivery 'not later than the end of May'.

The original promise in the order form, 'please allow 28 days from receipt of order for delivery' has therefore been multiplied by a factor of four. Careful examination of the latest adverts to appear in the computing press reveals that the company has now come clean, and makes only a promise of immediate acknowledgement with an expected shipment date.

The press are as badly off as the punters: not a single review sample has been received by any magazine and so the review we intended to publish this month will have to be postponed. As a result no independent source has had an opportunity to examine Sinclair's claims for the QL, yet the company has managed to pick up 500 orders a day since January 12, that is 30,000 in the first two months, worth some £12 million.

Meanwhile an industry of paper tigers is growing to complement the elusive paper machine. The announcement of the first 'independent QL users' group' came within two weeks of the launch. A six month trial subscription to IQLUG of £3.25 purchases access to a 'free software library' (not too well stocked at present one suspects), advice, workshops, support, and a monthly newsletter. The latter contains such gems as a terminal program written for the TRS-80 II running C which 'should work almost "as is" on the QL when the C compiler comes along'. Over extended foresight at work here.

Joe the Lion software have already announced a Spectrum emulator for the QL, which also, unfortunately, is not quite absolutely completely ready yet – Joe needs a team of programmers to write the coding stage, all-comers apply to Lawrence Holt on 061 366 5935.

The Emulator has been given an expected price of no less than £25, a worrying indication of the likely cost of all independent QL software. The high cost is due to the Sinclair monopoly on microdrive tapes and the necessary inclusion of a connector (see letters page for more on this).

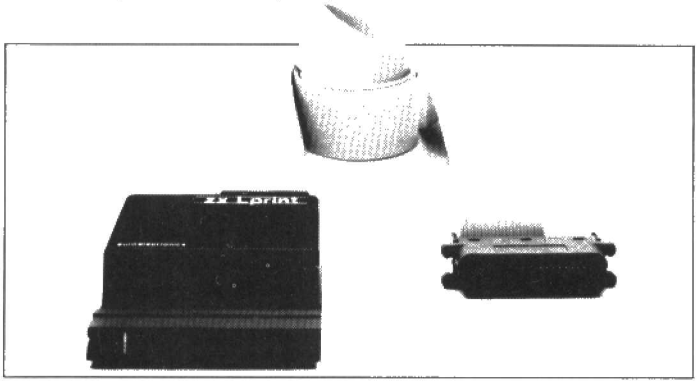
Last but not least of the QL bandwagon leapers is our own dear publishing industry. Already two QL Users have appeared within existing computer magazines, and a rather bedraggled 3 page 'Sinclair Business User' in another.

Breaking down the language barriers

Computers of different type have extreme difficulty in conversing, as we all know, so Radio 4's new computer magazine 'The Chip Shop' has overcome the problem of broadcasting programs over the airways to a variety of machines by employing a utility known as Basicode.

Basicode was developed by a group of Dutch amateurs two years ago, and has since been used by the Dutch Broadcasting service (NOS) for the transmission of software by radio. The system is a standard electronic code which is entered as cassette software into the user's computer, and converts signals transmitted in Basicode into the computer's own BASIC. The system can also be used to transmit. Once the Basicode is entered programs can be transmitted to any one of a range of other micros; these include the BBC micro, Commodore 64, Pet and Vic 20, Sharp MZ80A, ZX81, Tandy TRS 80, Video Genie, and Apple II and IIe.

The kit, which includes cassette and handbook, is available from Basicode, Broadcasting Support Services, PO Box 7, London W3 6XJ and is priced at £3.95.



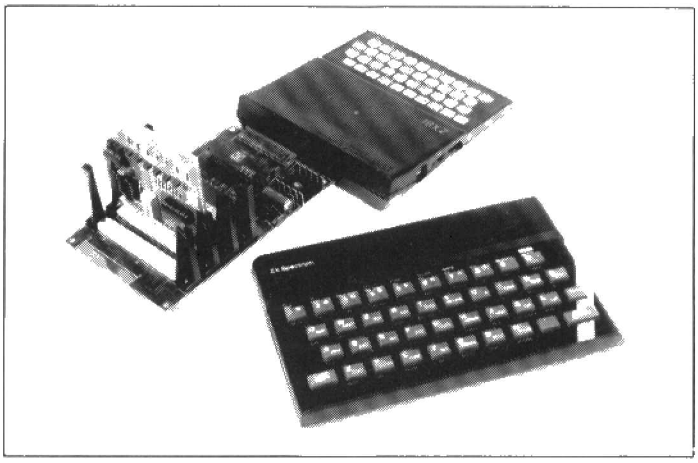
Two in one printer interface

Euroelectronics, manufacturers of the successful ZX Lprint interface have released a new version, the Lprint III which includes RS232 and Centronics ports on a single interface.

Biggest advantage of the new unit is the COPY command by which high resolution screen dumps can be made to a number of popular printers without the need for any extra software; these include Seikosha, Epson, Star, Walters, Shinwa and CTI models.

ZX Lprint is fully compatible with Interface 1 and Microdrives, and with 'a popular Spectrum Prestel adaptor' (text or full graphics Prestel pages may be printed, even in colour).

Lprint is priced at £34.95 plus £9.95 for either Centronics or RS232 cable. Euroelectronics are at 26 Clarence Square, Cheltenham,



Two new motherboards (pictured above) one for the Spectrum and one for the ZX81, are available from Velleman. The boards offer space for up to four interface cards, and are provided with a 23 pole edge connector at the back, giving the facility to connect the ZX printer or to stack more motherboards one after the other (known as backplaning out of the window). Power is taken from the computers' 9V DC supply or (more sensibly) from an external unregulated 9-12V DC supply, depending on the consumption and number of cards used. Each system can be used to add the following features: eight open collector outputs; A/D conversion; eight optocoupler inputs; Centronics interface; and D/A conversion. Several other cards are being developed.

BYTE BACK

Send your letters to The Editor, E&CM, 155 Scriptor Court, Farringdon Road, London EC1R 3AD.

Sir,

With reference to your review of the new Sinclair QL computer I have to disagree with your reviewers comment about the QL not requiring an ordinary cassette interface.

On the contrary, unless your reviewer wishes to see the price of software increase by the use of a dedicated source of program medium such as the Microdrive, because nobody is, at the present time, offering duplication facilities to software houses. I therefore feel that his comments are quite untrue because Microdrive cartridges are at the moment £4.95 each, so increasing the cost of any software released on them whilst the cost of duplicating a piece of software on cassette can cost as little as 40p per cassette.

The lack of an ordinary cassette interface and Centronics interface are, I feel, two major mistakes in what appears to be a marvellous machine by Sinclair, but only time will tell whether I am right or wrong.

G. Cradle
Porthcawl

Firstly, the Centronics port was a curious omission from the QL: most printers use parallel ports as standard, providing RS232 as an optional extra. Anyone already possessing a Centronics type printer will have to go to the expense of modifying it to work with the QL. This point was made in our review.

Sinclair's attitude to the Microdrive will have to change if successful software development for the QL is to take place. As you say there is only one facility in the world for the manufacture, and more importantly, the duplication of Microdrive cartridges: Sinclair's. Further, the cartridges wear out after about six month's intensive use and no software house in its right mind would (if it could!) protect its cartridges to prevent copying because they would become useless so quickly. Fortunately it would be a simple matter to add a cassette interface (so why didn't Sinclair?).

For early QL users life will not be as easy as it could be, and we take your point about the cassette interface (or lack of it). (Ed.)

Sir,

I was very pleased to see the "68705 Cross Assembler" described in the January issue. Unfortunately, the listing which ends at line 18040 is incomplete (eg there is a "goto 18050" earlier). It looks as though you have forgotten to print a page. Can the missing list be provided, please.

With reference to your interest in the 68705 microcontroller, I draw your attention to the MC146805 which is a CMOS microprocessor with exactly the same opcodes. This particular device (which contains a timer and two parallel ports) enables portable low-current battery-operated equipment to be made. For example, I have made a small computer for my local Hospital which is strapped to the body and takes measurements of skin temperature at regular intervals over many hours. The data collected is then passed via parallel ports into a BBC micro for processing.

I would like to rewrite the 6805 cross-assembler for the BBC-micro to use in this work. Hitherto, I have written in Assembler and translated by hand into machine code. but a complete listing is needed!

Dr. J. R. Barker

EDITOR'S NOTE
The following are the 3 final lines of the assembler, which were unfortunately omitted.

```
18050  LET R$=R$=I$
18060  NEXT Y%
18070  RETURN
```

Sir,

I feel I must reply to the comments made by Mr. Martin Ridout of Picturesque regarding my review of that company's assembler and monitor. There were no errors of fact in my review – I pride myself in checking any contentious points I may make, and I will therefore take each of Mr. Ridout's points in turn.

1) Hisoft's advertising is, like any responsible advertiser's should be, based on conservative estimates taken on a worst-case basis. In fact, I have it on reliable authority that the quote of 3000 lines assembled per minute was based on a disc-based assembly (thus including disc accesses) of Hisoft's Pascal compiler, consisting of 11000 words of code. I defy Picturesque to beat that under similar conditions. If Devpac was presented with a mere 1000 lines of code, it too could assemble at the rate of 4500 lines per minute.

2) Although the price difference between Devpac and the comparable Picturesque products is only £2.00, the 'far, far cheaper' quote came about because the Hisoft product offers rather more.

3) The non-presence of ASCII characters from the keyboard may well be my error, but perhaps this is a reflection in the documentation, Mr. Ridout?

4) In no way is the Picturesque assembler transparent to the user (and this also applies to the disassembler/monitor) – restrictions are undeniably placed upon the user if he has to issue instructions to the assembler to assemble from one location code that is written to another. Plus the fact that the size of source files are limited by memory. This is not the case with Hisoft's product.

5) The rest of Mr. Ridout's letter goes on to extoll the virtues of products that haven't even been launched yet. I find it hard to take seriously the complaints from someone who relies on promises to cover up for inferior product.

Thank you.

Adam Denning

Dear Sir,

Thank you for the opportunity to comment on Adam Denning's reply to my letter following his review of our Spectrum Assembler and Monitor.

It now seems from Adam's reply, that the assembly facilities of our cassette-only compatible Assembler were being compared with a system capable of supporting assembly from discs, and as such, it was not a fair comparison. As machine code programming is becoming much more popular with home computers owners, perhaps it is time that a form of Benchmark test, similar to those used in comparisons of other languages, should be introduced to clarify relative claims and to enable reviewers to make comparisons on equal terms.

Assembly speed is undeniably important, but so is the speed and **ease** of use of the overall system. Total flexibility, which Hisoft's system appears to offer, can cause problems particularly for less experienced programmers, as it must complicate the use of the programs. I do accept that our system does impose some limitations, but these have been minimised as far as possible, and I believe that the limitations that do exist are an acceptable trade-off against increased "user friendliness".

But this does not mean that our products are only of value to beginners. Many professional and freelance programmers regularly use our Assembler and Monitor to write commercially available Spectrum software. Surely they would not use utilities that imposed severe limitations.

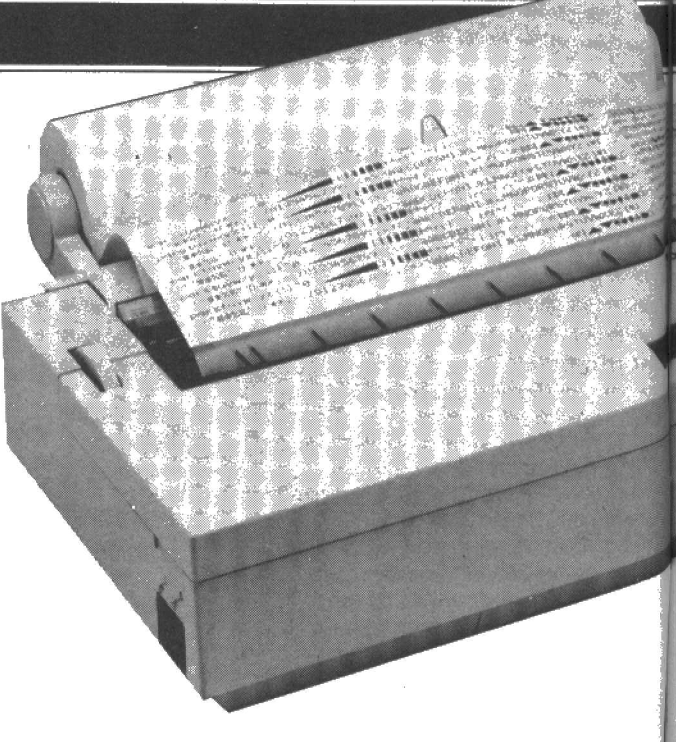
Finally, I totally refute the suggestion that I was trying to cover up an "inferior" product with promises of improvements. I am proud of the programs that we market, and of the reputation that they have earned among experienced programmers, and my reason for referring to the new programs was to indicate that we are constantly seeking to improve the facilities in our programs.

Martin J. Ridout
Picturesque

This letter has been shortened

It is customary at this point to announce that all correspondence on the subject is closed, however Mr. Ridout's point about m/c benchmarks is of interest: does anyone have any suggestions? (Ed.)

Centronics



Printing is timewasting – unless you have a buffer. Richard Harvey's design for frustration free computing occupies no machine memory and is completely transparent to the user – for only £25.

With the current low memory prices it is now possible to design and build stand-alone microprocessor based devices which have large data storage capacities. One such device is this Printer Buffer. At a total cost of around £25, the design provides by far the cheapest way to obtain a buffer, a very useful device to anyone using a computer who needs to print large quantities of text. The buffer allows the computer to be used whilst the printer is printing. It is connected between the computer and the printer using the Centronics standard parallel printer interface and thus can be connected between any computer with a centronics type printer port and any printer with a centronics type input port. Almost all the widely used printers have a centronics input.

Anyone printing a letter on a slow letter quality printer, or a large source file on any standard printer would normally expect to wait some time before using the computer again. The printer buffer holds text while it is printed and frees the computer for other work.

The computing power required to print text is fairly minimal and once formating has taken place it merely involves sending the text, character by character to the printer and waiting while the printer is busy. This is slow by nature of the fact that printers generally only accept data one character or one line at a time, and print this quite slowly making the computer wait while they do so. The amount of time it takes to send data down a parallel interface (if no waiting occurs) is negligable in comparison. The buffer receives the text from the computer very quickly and rapidly begins to fill its memory but, simultaneously the buffer also starts sending the data to the printer but at the slower rate it can accept. This means that the buffer must hold each character received until it is its turn to be sent to the printer. It follows then that if the text is smaller than the buffer's memory size then the computer will be free immediately. If the text is larger than the buffers memory size then the user must wait until the text left to print becomes smaller as it is printed out.

Elastic buffers

In order to perform this task the buffer requires something known as an 'elastic memory' or more formally a FIFO (First In First Out) memory. FIFOs can be implemented in two ways: either using a hardware FIFO memory device or by simulating one using a microprocessor.

Hardware FIFOs are generally quite expensive, especially if a large capacity is required and it would require a lot of support chips to implement a complete printer buffer using one. They are generally used in very high speed applications.

Microprocessor simulated FIFOs are very easy to implement and can use cheap memory devices so they can have very large capacities. Also, by using a microprocessor, the logic for the printer buffer becomes mostly software and thus the total unit can use very few chips. For this reason the microprocessor alternative was considered the only rational method of constructing a printer buffer. Later on we will see how the FIFO is implemented, but first let's examine this project from the hardware point of view.

Choice of processor

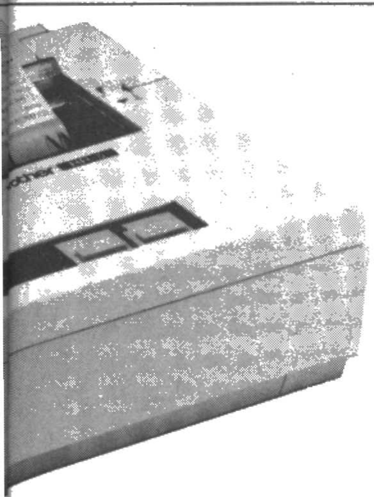
There are many microprocessors on the market. Most are intended for use with other support chips although there are a growing number which are designed with stand-alone applications in mind (applications similar to this), some common types are the Z8 and 68705 etc. These stand-alone versions are known as single chip microcontrollers and are appearing in many applications which previously required boards of TTL to perform the same function.

This design uses the Z80 microprocessor which is not a single chip controller and at first it seems an illogical choice mainly because it is usually associated with home and business computer systems, but I will show that it was the ideal processor for the job and I contest that the printer buffer would have been much harder to implement using a single chip controller. I am not saying that the Z80 is superior to single chip controllers (far from it!) but in this application it has some definite advantages.

Memory requirements

The printer buffer requires by definition to contain a lot of memory. This memory must be quickly accessible and so must be solid state. We could use Static Random Access Memory (SRAM); these can be connected very simply to most processors. Currently the highest capacity chip is 8KBytes which is about the chip density we require, but this is very expensive and still not generally available. The next size down is a chip with 2KBytes, but for a 64K byte buffer we would require 32 chips. This would occupy a large amount of board space and cost around £100!

The cheapest high capacity memory available today is Dynamic Random Access Memory (DRAM) but it has the drawback that its contents must be refreshed every 2mS (more on this later). Refreshing generally involves extra hardware, this is unless the processor used can help. The advantages of DRAMs though are staggering, a 64K DRAM chip can hold 8K bytes of data and only costs about £5. So the eight chips needed to give 64K bytes would only cost £40.



printer buffer

We must then use DRAMs if we are to make the device cheap and compact and as we will see later the amount of DRAM the user can add is variable.

Refreshing memories

Externally a DRAM looks much the same as any other memory; it has data inputs and outputs, a write enable line and address inputs. The DRAM used in this project has multiplexed address inputs; that means that two sets of address lines are put onto the same pins. This saves pins on the chips thus allowing a 64K bit DRAM to be put into a 16-pin DIL package. A 64K DRAM requires 16 address lines to uniquely select any memory location and as the address lines are multiplexed, the top eight address bits are passed in on the same pins as the bottom eight address bits. To distinguish them the bottom eight bits of an address are known as the Row address and are presented to the DRAM first during a memory cycle, and the top eight are called the Column address and are presented next. So that the DRAM knows what is going on, two inputs: RAS and CAS are provided. Whilst the Row address is presented to the DRAM the Row Address Strobe (RAS) is lowered and when the Column address is presented the Column Address Strobe (CAS) is lowered. Both RAS and CAS are raised at the end of the memory cycle. **Figure 1** shows the relative timing of these signals.

'. . . the printer buffer holds text while it is printed and frees the computer for other work'

The above was a potted account of how a DRAM is addressed and is necessary to explain how refreshing is performed. One ROW of the DRAM is refreshed by simply presenting the Row address and dropping RAS, no CAS need by supplied, and the DRAM does not output any data (see **Figure 2**). All the rows in the DRAM must be refreshed at least every 2mS or data loss will occur. Incidentally, refreshing also occurs during normal memory accesses because it is initiated by supplying an RAS signal and the fact that a CAS signal is also supplied has no bearing.

Thus our hardware must generate sequential refresh cycles and must interleave them with normal memory access cycles, in addition to which it must multiplex the address bus and supply RAS and CAS signals!

By now you are probably thinking: Come back SRAMs all is forgiven! But fear not, this is where the Z80 comes into its own. The Z80 contains the logic necessary to generate sequential refresh addresses and will generate a refresh cycle every time an opcode fetch is performed. What's more, it allows the programmer to alter the value of the refresh address as it is held internally to the CPU in the R register, it will become apparent why we need this later.

Multiplexing

The other drawback with DRAM is the need to multiplex the address inputs, **Figure 1** shows the timing diagram of a memory access with multiplexing. The Z80 has an addressing range of 64K and thus has a sixteen bit address bus, so we could use multiplexer ICs and convert this to an 8-bit multiplexed address bus compatible with the DRAM. We would also need to generate the RAS and CAS signals thus requiring more external logic. The logic required to do all this is quite cheap but it makes the

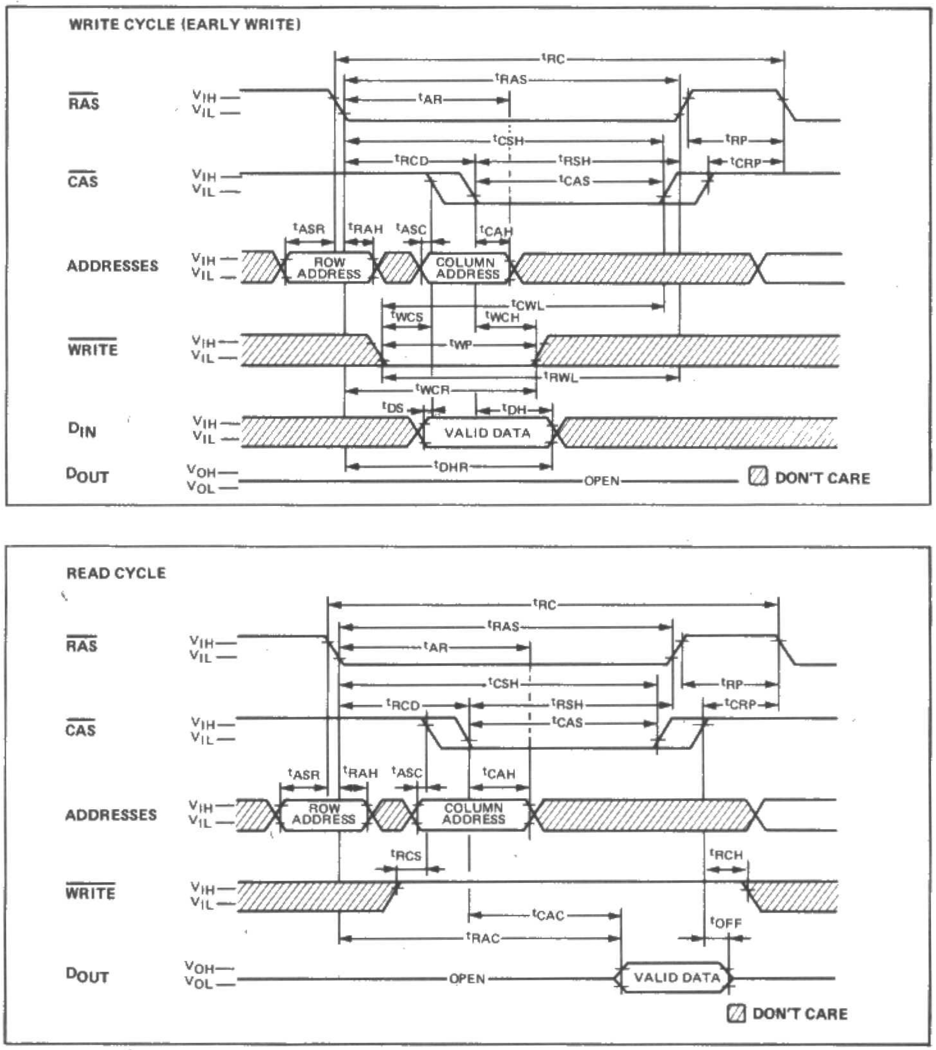


Figure 1. Timing waveforms.

circuit big and inelegant and if the design was intended for mass production it would be unnecessary expenditure if a better solution was possible.

Is there another solution? . . . Well, yes there is and all thanks to another aspect of the Z80's refresh hardware!

As I said earlier the refresh address supplied to the DRAM can be changed by altering the contents of the R register in the Z80, so perhaps this could be somehow used to implement multiplexing. **Figure 3** shows the way the Z80 generates refresh cycles and **Figure 4** shows a memory access as performed by a LD (HL),A instruction.

We will now examine the sequence of events that occur during the instruction: LD (HL),A (This instruction takes the contents of the accumulator and puts it at a memory location whose address is in the HL combination register).

The sequence of events in the execution of this instruction are very close to that required by a DRAM memory access, but with some modifications: firstly the instruction opcode is fetched, then the CPU (whilst processing the opcode) generates a refresh address which appears on the bottom eight address lines; finally, the memory access is performed and the least significant byte of the memory address appears on the bottom eight address lines. So all we need to do to eliminate the need for multiplexers is to connect the DRAM to the bottom eight address lines directly, then arrange to generate a RAS strobe when the refresh address is presented and if we intend to access the DRAM a CAS strobe is generated when the following memory cycle occurs. Thus the DRAM would write data from the data bus into the addressed memory cell. This would mean that the row address would be whatever was in the R register prior to execution of the LD (HL),A instruction, and the column address would be whatever was in the L register! The top eight bits of the memory address are arranged to determine whether the DRAM is accessed (ie: if CAS is dropped) creating a mechanism of generating a multiplexed address for the DRAM without multiplexers and with the very minimum of external logic. This system will also mean that the DRAM will be refreshed automatically as well.

I hope this illustrates the concept that in certain cases software can replace hardware. You may have realised that the software required to access the DRAM will be very involved. As a rule, this method would be totally inappropriate for home computers, but, it is ideal for the printer buffer where memory accesses are relatively infrequent by comparison.

The code to read a byte from the DRAM would be:

```
;A = msb of address, l = lsb of address,
H = DRAM select
    LD    R,A
    LD    A,(HL)
```

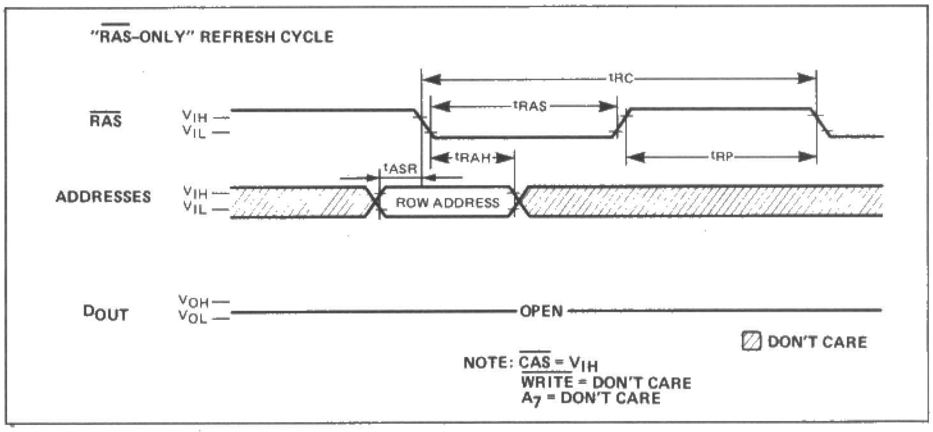


Figure 2. "RAS-ONLY" refresh cycle.

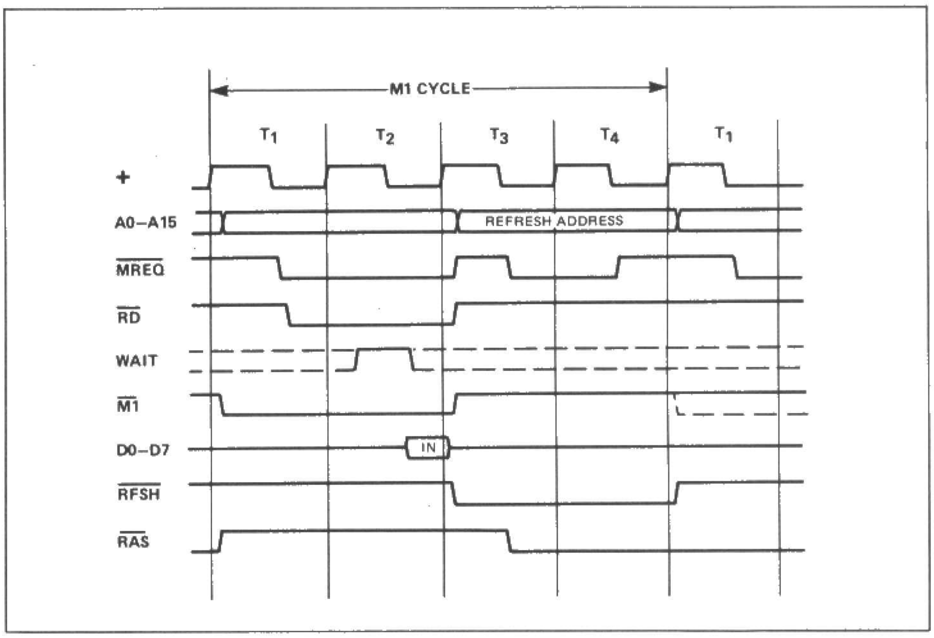


Figure 3. Instruction op code fetch for the Z80.

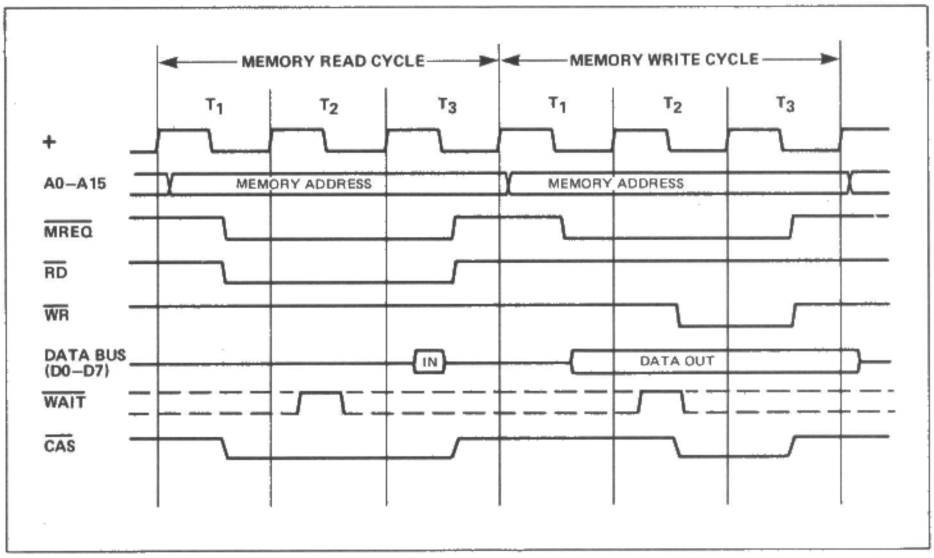


Figure 4. Memory read or write cycles for the Z80.

It is worth preserving the contents of the R register so that refreshing is not adversely affected. Steps are also taken to provide a full eight bit refresh address, because the Z80 only counts on the bottom seven bits of the R register. This is required so that 64K DRAMs requiring 8-bit refresh addresses can be catered for (some of the older versions only needed 7-bit refresh addresses and would work on the Z80 without any trouble).

Next Month: the PCB foil layout and software.

Spectrum TRACE

This program provides the Sinclair Spectrum with one of the most useful debugging aids in a Basic interpreter. A continuous display of the line numbers being executed is displayed. Nicholas Ryman-Tubb is the author.

After reading the excellent article on interrupts in the December, 1983 issue of *E&CM* I put two and two together and came up with a versatile machine code trace program.

The program will allow the trace function to be switched on and off by simply changing a variable. The speed of execution can also be changed to allow the line numbers to be displayed slowly.

As each line is executed its line number is displayed in decimal at the top right of the screen, in inverse. Any BASIC program can be run under the trace system: there is no corruption of the screen except for the top right four bytes.

Interrupting!

The Spectrum usually uses interrupt mode 1. This is where the processor always jumps to 38H (executes an RST 38H). At this address the keyboard is scanned and the frame counter updated. However, to allow the line numbers being executed to be continuously displayed, it is necessary to re-vector the interrupt. This is done by changing the I register to point to the page of the interrupt vector (MSB). The interrupting device (—DMA controller within the Spectrum hardware) supplies the LSB (which is OFFH), so the address of the new interrupt service routine (IRS) must be stored at I+OFFH, in this case I=OFEH. Once the vector has been set up a return is made to the system.

Then, when an interrupt occurs (every 20ms normally) a jump is made to the new ISR. This routine *must* execute an RST 38H at some time or the system will 'hang-up' (because the keyboard is not scanned).

The LSB of the vectors address is not supplied by the DMA controller directly but because of the fact that the data lines are floating and so have the default value of OFFH.

The interrupt service routine

Upon entering the ISR an RST 38H is executed, this scans the keyboard and keeps the ROM happy!

The next part of the program searches for a variable called 'trace'. If this variable has the value zero or does not exist, then the routine is terminated and no trace occurs. If its value is 1 then the trace is activated. If the value is −1 then the program deactivates mode two interrupts and resets the system to mode 1.

If the value is 1 then the program checks to see if a program is being executed. This is done by checking the high byte of the system variable PPC (23621). If this is set (255) then no program is being executed and the ISR terminates.

'the trace can be turned on or off and execution slowed'.

The next part of the program does a check on the line number being executed and the line number which was executed at the last interrupt. If the two are the same (ie the interpreter has not executed one whole line between successive interrupts, about 20ms), then the routine is ended.

Next the program checks the variable 'speed'. This variable, when set, controls the speed of execution per line. In fact it controls the delay between two successive lines and so some line numbers will be displayed quicker than others. If the variable is not set or set to zero then there is no delay between each line and execution continues at a maximum rate.

Finally, the line number is converted to decimal (using the same method as in the ROM at address 1A1BH). It is then displayed on the screen in inverse at the top left hand side, as four digits with leading zeros.

It must be remembered that all registers *must* be preserved during the execution of the ISR. This is the reason why most of the ROM routines cannot be used since their use may change the internal system variables, and the system would crash. This is why the ISR contains its own hex to decimal and printing routine.

Entering the program

A Basic program is given to allow the program to be entered as hex. The program will then be saved on tape as machine code, by typing SAVE 'trace' CODE 65000,212.

Once the Basic program has been typed in and saved you can enter the codes found in the assembly listing. The program will not allow any other characters than 0-9 and a-f. Two digits must be typed (eg 09,55,00). After each code has been entered push enter. When all the codes have been entered type STOP (SYMBOL SHIFT+A) to exit.

Running the program

Once the program has been entered the command: 'RANDOMIZE USR 65000', should be entered. This changes the interrupt vector. Any BASIC program can now be traced.

Hex loader

```
10   REM******************************
20   REM*   HEX LOADER PROGRAM       *
30   REM*Copyright Tubb Research,1983*
40   REM******************************
50   CLS
55   REM ** Get start address of code **
60   INPUT "Start address";Add
70   LET a$=""
80   PRINT INVERSE 1;AT 0,0;"ADDRESS";AT 0,15;"DATA"
85   REM ** Number of lines per page **
90   FOR Y=2 TO 20
100 PRINT AT Y,0;Add
105 REM ** Display the cursor **
110 PRINT AT Y,(15+LEN a$); INVERSE 1; FLASH 1;"?"
120 IF INKEY$<>"" THEN GO TO 120
130 IF INKRY$="" THEN GO TO 130
140 LET b$=INKEY$
150 LET t=CODE b$
160 IF t=12 AND LEN a$=0 THEN LET b$="":GO TO 150
170 IF t=226 THEN STOP
180 IF t=12 THEN LET a$=a$( TO (LEN a$-1)):LET b$="":
PRINT AT y,(16+LEN a$);" "
190 IF LEN a$>=2 THEN LET a$=a$(1 TO 2):LET b$=""
200 IF t<48 OR t>102 THEN LET b$=""
210 IF t>58 AND t<97 THEN LET b$=""
220 LET a$=a$+""
230 PRINT AT y,15;a$
240 IF LEN A$<2 THEN GO TO 110
250 IF T<>13 THEN GO TO 110
260 LET t=0
270 FOR x=1 TO 2
280 LET a=(CODE a$(x))-48
290 IF a>9 THEN LET a=a-39
300 IF x=1 THEN LET a=a*16
310 LET t=t+a
320 NEXT x
330 PRINT AT y,17;" "
340 PRINT AT y,20;t;"D"
350 POKE Add,t
360 LET a$=""
370 LET Add=Add+1
380 NEXT y
390 CLS
400 GO TO 80
```

The commands are:

LET trace=1	Switch the trace on
LET trace=0	Switch the trace off
LET trace=−1	Revert to normal Spectrum interrupts
LET speed=0	Maximum execution speed
LET speed=255	Maximum delay between lines (slow execution)

These commands can be entered within a program, to trace all or only part of it, or immediately – but remember to GOTO the line number! If the speed variable is not specified then maximum execution occurs. An example program is given on the right.

To revert to normal Spectrum interrupts use the command: LET trace=−1.

This need only be done around the INKEY$ statement, for example:

```
10   LET trace=−1
20   IF INKEY$="" THEN GOTO 20
30   RANDOMIZE 65000.
```

■

Spectrum trace program

```
        (FDE8)
FDE8   21F7FD
FDEB   22FFFE
FDEE   F3
FDEF   3EFE
FDF1   ED47
FDF3   ED5E
FDF5   FB
FDF6   C9


FDF7   FF

FDF8   F5
FDF9   E5
FDFA   21B4FE
FDFD   CD89FE
FE00   B7
FE01   286C
FE03   3C
FE04   CAA9FE
FE07   2A455C
FE0A   24
FE0B   2862
FE0D   25
FE0E   C5
FE0F   D5
FE10   E5
FE11   ED5BC0FE
FE15   ED52
FE17   E1
FE18   2853
FE1A   22C0FE
FE1D   E5
FE1E   21BAFE
```

```
FE21   CD89FE
FE24   B7
FE25   2809


FE27   47
FE28   C5
FE29   0600
FE2B   10FE
FE2D   C1
FE2E   10F8


FE30   E1
FE31   AF
FE32   47
FE33   4F

FE34   11E803
FE37   ED52
FE39   3C
FE3A   30FB
FE3C   19
FE3D   3D


FE3E   E5
FE3F   CD72FE
FE42   E1

FE43   116400
FE46   AF
```

```
FE47   ED52
FE49   3C
FE4A   30FB
FE4C   19
FE4D   3D


FE4E   0E01
FE50   E5
FE51   CD72FE
FE54   E1


FE55   110A00
FE58   7D
FE59   93
FE5A   14
FE5B   30FC
FE5D   15
FE5E   83
FE5F   67


FE60   E5
FE61   0E02
FE63   7A
FE64   CD72FE
FE67   F1


FE68   0E03
FE6A   CD72FE
```

```
FE6D   D1
FE6E   C1
FE6F   E1
FE70   F1
FE71   C9


FE72   211C40
FE75   09
FE76   87
FE77   87
FE78   87
FE79   EB
FE7A   4F
FE7B   21803D
FE7E   09


FE7F   0608
FE81   7E
FE82   2F
FE83   12
FE84   23
FE85   14
FE86   10F9
FE88   C9


FE89   E5
FE8A   2A5D5C
FE8D   E3
FE8E   225D5C
```

```
FE91   D5
FE92   C5
FE93   DDE5
FE95   CDB228
FE98   DDE1
FE9A   C1
FE9B   D1
FE9C   3E00
FE9E   3804


FEA0   23
FEA1   23
FEA2   23
FEA3   7E
FEA4   E1
FEA5   225D5C
FEA6   C9


FEA9   F3
FEAA   3E3F
FEAC   ED47
FEAE   ED56
FEB0   E1
FEB1   F1
FEB2   FB
FEB3   C9


FEB4   74/72/61/63/65/OD
FEBA   73/70/65/65/64/OD

FEC0   (0002)
```

A bigger brother

The Brother EP44 is a high quality dot-matrix printer; it is also a typewriter; it is also a computer terminal or data transmitter. The EP44 cannot make the tea, but then nobody and nothing is perfect.

The entry of full computing facilities into the home has been retarded by the high price demanded for quality printers. In January we reviewed something of a breakthrough in this field, the Brother EP22. The EP22 scored on price, at £170 for a printer/typewriter, but not on quality. In particular the dot matrix print was crude, with a matrix resolution of 5 x 7; the editing facilities were limited (yes, editing facilities on a printer) and the memory too small.

By a strange coincidence, every niggling complaint made in that review has been answered by Brother in a new machine, the EP44. The 44 not only prints and types; it also sends data via the RS232C terminal located at the side of the machine. It could therefore be used as a dumb terminal for any computer which accepts standard ASCII codes (the horrible keyboard of the Spectrum would be a prime candidate for replacement) or to send information via a modem from anywhere in the world to anywhere in the world.

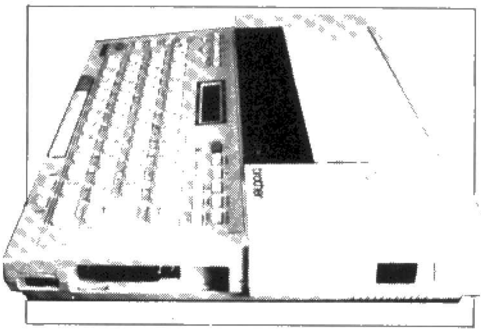
A speedy examination of the specifications reveals a performance in the order of double that of the EP22 – fitting in view of the part number designation. Firstly, the print resolution is very fine: the matrix is 24 x 18, giving a letter quality seriffed face. The EP44 is however limited in the weight of paper it can use – heavy cartridges gives a faded imprint. On light continuous feed paper or photocopy paper the resolution is excellent. A further deficiency of the EP22 – the 75 character width – has been remedied. The EP44 has a maximum width of the computer standard 80 characters. Finally, the range of available baud rates has been extended. Data can be received or transmitted at 300, 600, 1200, 74 or 110 baud.

Full marks then and a firm recommendation of the EP44 as a stand alone printer for use alongside any home computer with RS232C interface. At £220, the EP44 is cheaper than most printers on the market and certainly of better quality than any machine under £200. A solitary reservation is that print speed is slow – slower than the EP22 – at 16 characters per second.

Everything that follows is an added bonus to the computer user: that is, the keyboard, text editor and terminal. These facilities are nevertheless worth their weight in gold and without the print capability would still be cheap at the price.

The keyboard is satisfying to use and fast. A wide range of mathematical and financial symbols are available, plus a calculator on which calculations can be performed within any text format. Characters are entered in any one of three modes: non-print (which is used when keying text into memory); correction print (16 characters appear on the LCD screen before they are printed) and direct print, which is a typewriter mode. A further mode available on the EP44 is LL – line by line – in which a full line can be entered, corrected, and then printed. Status information on editing facilities, memory and batteries is constantly available on the LCD.

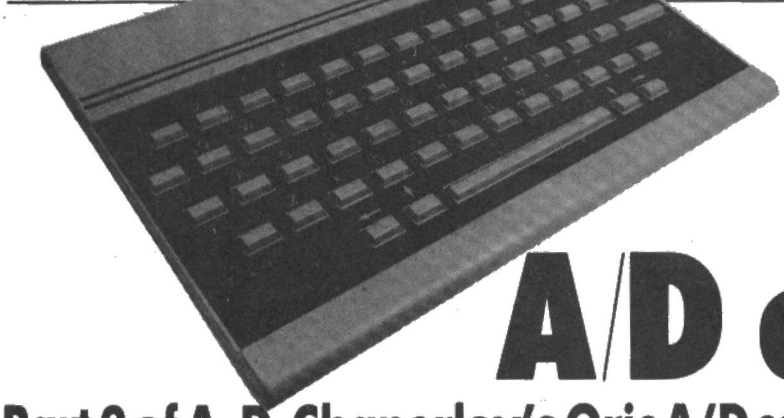
The text editor is obviously not up to the standard of a word processor: it cannot change margins once they are set, change the order of paragraphs or lines. It can however do a great deal. Up to 4000 characters can be entered into the memory, an adequate amount. It is possible to examine the text line by line and character by character, insert paras, lines, words and characters, make deletions, and add to the end of the text. Printing of text held in memory can be stopped and variables such as names and addresses inserted. There is a useful auto return and underlining facility during text entry, and lines can be centred or ranged flush to the right hand margin. Line justification is not available, which is a pity.



A side view of the EP44 showing the RS232C interface socket.

A few final points to clear up. The EP44 is very similar in appearance to the EP22. The case is cream instead of black, and the machine is very slightly larger, but still speaks the last word in portability, weighing in at 2.5kg. The EP44 is battery powered with a 6V DC mains source option. The manual is thorough and detailed, with a comprehensive technical appendix of control and data codes. A supplement gives connection details for a variety of US and Japanese computers, including the Apple II, Atari 400/600, Commodore 64 and VIC-20, TI99/4A, TRS-80 and the NEC PC range. Adequate pin-out data for the RS232 interface is given and it would be a simple matter to make the connection to a BBC micro or Spectrum (with Interface 1) for example.

The Brother EP44 is an excellent, well-designed and superbly versatile machine. It has no significant defects and many powerful advantages over the competition (of which the nearest is a poor man's version, the Silver Reed) and it is thoroughly recommended to anyone yet to buy a printer for their computer.



Oric A/D converter

Part 2 of A. D. Chanerley's Oric A/D converter project explains the construction and software.

The ORIC has only one power rail at 5V, this is used to provide the power to the ADC chip and also to provide the −10V reference by using a multiplier chain of capacitors and diodes. This provides, in theory a voltage triple the size of the supply, but taking into account voltage drops across the diodes this gives about 12V. The CMOS gates are connected in parallel and so increase available current. The CMOS Schmitt inverter gates, a 40106B, or a 7414 TTL type, are self oscillating and provide the square wave to drive the chain of capacitors and diodes. A 10uF tantalum capacitor smoothes the output and the 10V Zener clamps it at −10V to provide the necessary reference voltage to the 7581 ADC chip. The 1.6MHz clock input to the 7581 can also be provided by one of the inverters not in the capacitor chain. All that basically happens in the capacitor chain is that each capacitor maintains a steady DC charge, and pumps charge onto the next one along the chain on alternate half-cycles provided by the square wave from the self-oscillating chain.

Decoder for the ADC

Decoding of the necessary memory locations to interface this chip is effected by using a 74LS138,3 to 8 line decoder. This chip has 3 inputs A, B and C, which select one of the 8 data outputs, 0 to 7 at the appropriate pins. The truth table is shown in **Figure 6** along with a diagram of the chip. There are also 3 data enable pins 4, 5 and 6. Pins 4 and 5 are enabled when low and 6 when high, all 3 pins when enabled simultaneously, then enable one of the decoded output pins which then is driven low providing the CS (chip select) signal for the 7581. Simultaneously the same chip select signal from the decoder also pulls low the I/O CONTROL line on the ORIC expansion socket, which, as mentioned previously is necessary in order to disable the ORIC internal 6522 VIA at the moment of data transfer. The locations chosen are from &03F0 to &03F7, 8 locations corresponding to the 8 analogue channels of the ADC. A0, A1, A2 are connected directly to the 7581 chip and decode the chips' internal RAM which contains the converted data byte from the corresponding analogue channel. A7, A6 and A5 are connected to the A, B, C inputs of the LS138 decoder and decode pin 7 which is pulled low when simultaneously A3 and I/O go low and A4 goes high, This occurs every time an address from 03F0 to 03F7 is accessed, as shown in **Figure 8.** The I/O line, pin (5) on the ORIC expansion socket decodes the upper address lines corresponding to A8 up to A15 as previously explained.

Software

The test software to test the ADC corresponds to the test circuit shown in **Figure 9.** Each analogue channel of the ADC can accept up to 10V input therefore a PP3 giving up to 9V is quite suitable. Its output being first divided by a 5k or 10k potentiometer. This is then fed straight into one of the ADC channels, which incidentally have an input impedance of 22k each. The software is as follows:

Test software

```
10  CLS
20  Y=PEEK(#03D0):REM read channel 0
30  PLOT 13,14,STR$(Y)+"":REM show data byte at grid 13,14
40  GOTO 20
```

The result of this software will be to continuously plot the value of the current data byte corresponding to analogue voltage being input into, in this case, channel 0. Any of the 8 channels may be selected corresponding to the assigned memory location between 03F0 to 03F7. The conversion relation is as follows:

V=kN

when V=10 Volts, the maximum input, N=255, the maximum value of the 8-bit word. Hence the value of k=10/255. Therefore for a new PP3, at 9 Volts, N should be 230. Initially, with no input voltage into the channels, each channel floats to zero, and therefore the converted data byte will read 0 at grid 13,14 on the VDU.

Construction

Since the 7581 is such a well integrated chip construction of the complete ADC around it is relatively straightforward. First solder in the 3-DIL sockets for the 3 chips. 28-way for the 7581, 14-way for the hex-Schmitts and 16-way for the decoder. The link wires which in this case are single stranded and insulated. Then solder in the 68pF capacitor and 680R resistor for the 1.6MHz clock, this can then be followed by the 4xIN4148 diodes, ensure the correctness of position for anode and cathode. The 3x47n ceramic capacitors, followed by the 10u tantalum bead capacitor – take due care that this capacitor is connected with correct polarity. Finally solder in the Zener clamper and the 22k feedback resistor. The ORIC has a 34-way male expan-

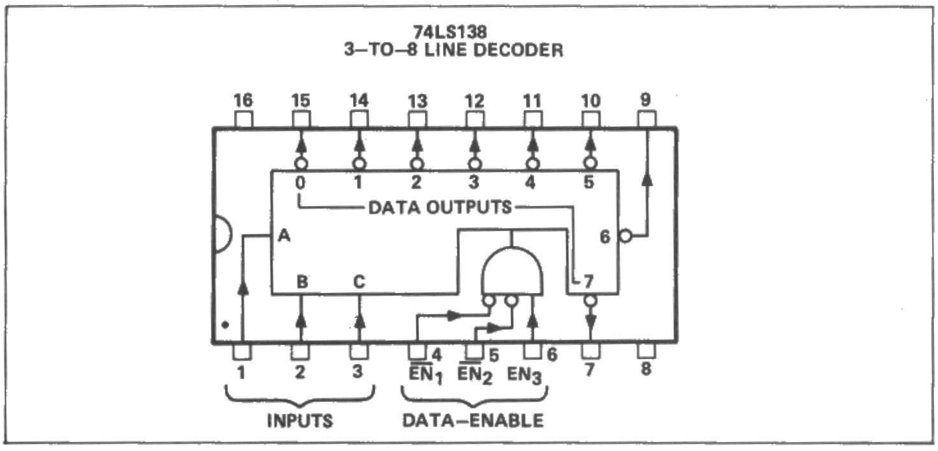


Figure 6(a). Pin out of the 3-to-8 line decoder.

ORIC-1 LOCATIONS	CHANNEL
#03F0	0
03F1	1
03F2	2
03F3	3
03F4	4
03F5	5
03F6	6
03F7	7

ORIC PIN OUTS

0V	34
+5V	33
D7	22
D6	14
D5	19
D4	18
D3	16
D2	9
D1	12
D0	10
A0	13
A1	15
A2	17
A7	25
A6	23
A5	21
A3	11
$\overline{\text{I/O}}$	5
A4	20
$\overline{\text{I/O CONTROL}}$	6

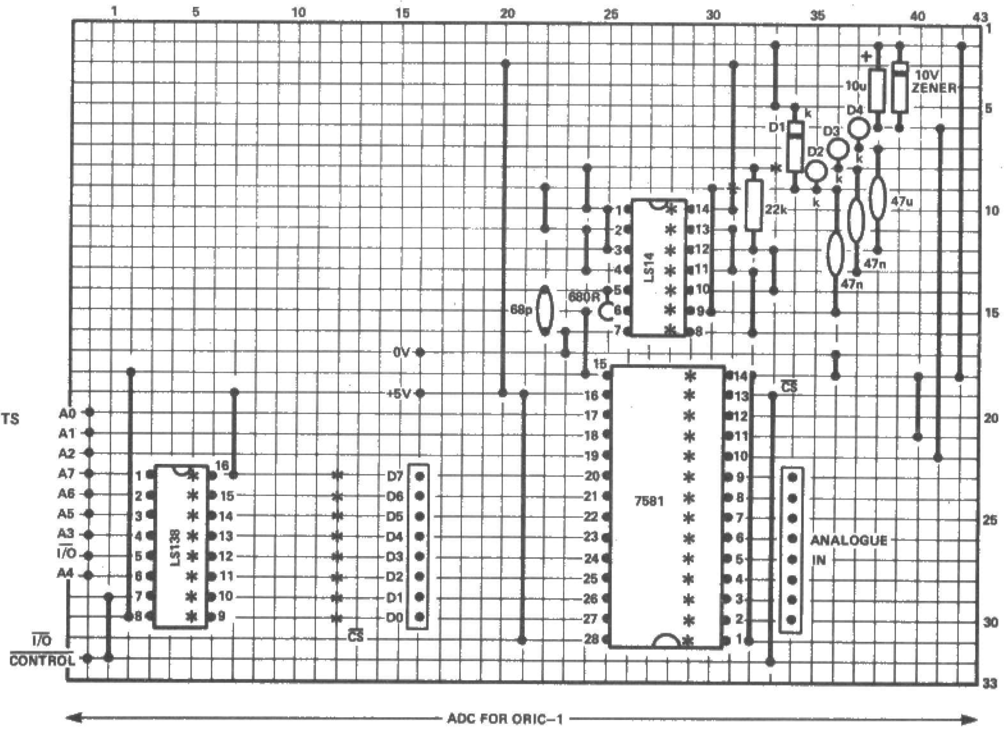


Figure 7. Vero board layout for the A/D converter.

TRUTH TABLE

$\overline{EN}_1$	$\overline{EN}_2$	EN_3	C	B	A	OUTPUT LOW ACTIVE
0	0	1	0	0	0	0
0	0	1	0	0	1	1
0	0	1	0	1	0	2
0	0	1	0	1	1	3
0	0	1	1	0	0	4
0	0	1	1	0	1	5
0	0	1	1	1	0	6
0	0	1	1	1	1	7

← THIS ONE USED IN ADC PROJECT

Figure 6b (left). Truth table of the 74LS138.

Figure 8 (right). Accesses to the ADC are in the range 03F0 to 03F7.

PAGE 3 LOCATIONS ADDRESS

A15.....A8	A7	A6	A5	A4	A3	A2	A1	A0	BITS
$\overline{\text{I/O}}$	1	1	1	1	0	0	0	0	03F0
$\overline{\text{I/O}}$	1	1	1	1	0	0	0	1	03F1
$\overline{\text{I/O}}$	1	1	1	1	0	0	1	0	03F2
$\overline{\text{I/O}}$	1	1	1	1	0	0	1	1	03F3
$\overline{\text{I/O}}$	1	1	1	1	0	1	0	0	03F4
$\overline{\text{I/O}}$	1	1	1	1	0	1	0	1	03F5
$\overline{\text{I/O}}$	1	1	1	1	0	1	1	0	03F6
$\overline{\text{I/O}}$	1	1	1	1	0	1	1	1	03F7
#03	F				"X"				

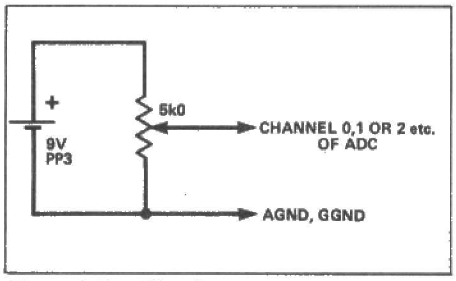


Figure 9. Test Circuit.

sion socket and so the corresponding female connector is needed with the speedbloc ribbon cable for easy insertion, without soldering, onto the 34 shearing pins.

Applications

Applications are many, ranging from joysticks to transducers. The latter will require an amplifier, for example a temperature transducer for continuous monitoring of one's bath water! Or an optical transducer used in conjuction with a burglar alarm, or for the measurement of rotational speeds using an interupted light beam or a magnetic sensor ■

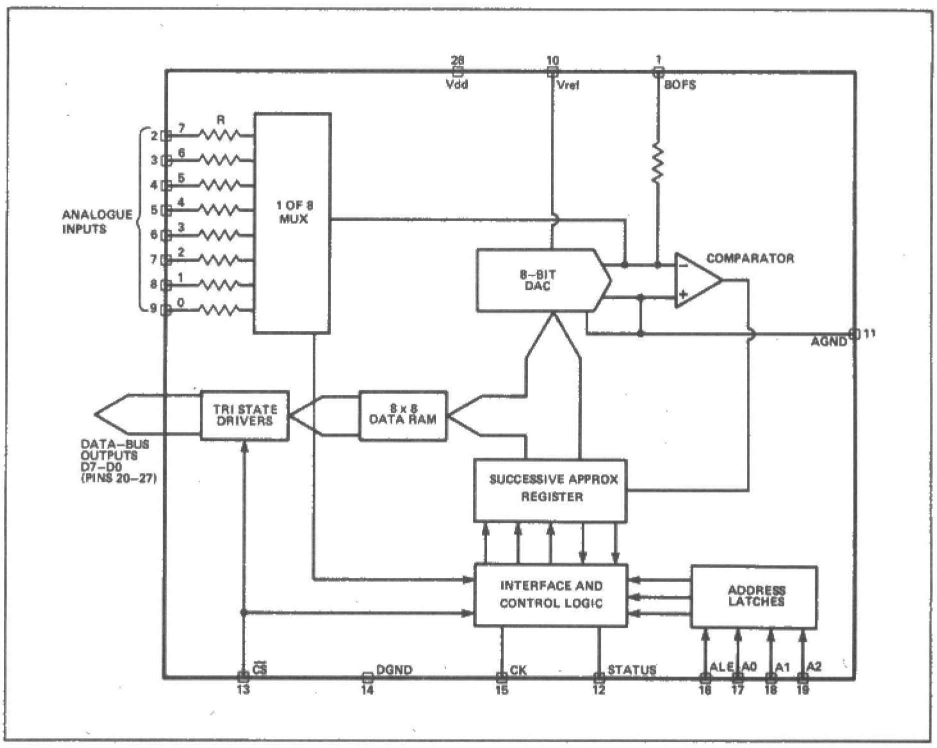


Figure 10. Block diagram of the ADC chip.

EXCLUSIVE BBC SOFTWARE

Graphics designer

Anyone using this teletext editor will be able to realise the full potential of the BBC B's Mode 7 graphics. Text and pictures as good as any seen on Prestel can be constructed. Adam Denning designed the program for Micronet 800 – this is an exclusive sneak preview for *E&CM* readers.

The BBC Micro's Mode 7 (Teletext) is the most memory efficient screen mode on the machine, yet its complicated set of control characters puts a lot of people off using it to its full potential. This program should help cure that.

Originally designed for Micronet and Prestel editors, the frame editor is a much slimmed down version of a more complicated and advanced editor that I wrote in BCPL. It has been converted to BBC Basic in such a way that the structure of the BCPL original has been maintained as far as possible, plus it has been given the benefits of being able to cater for all the BBC's filing systems – cassette, disc and Econet. The original only needed to cater for discs.

Due to this multi-filing system capability, disc users will find sections of the program to be slower and less efficient than they need be, and at the end there are some tips on how to regain the speed lost in catering for cassettes.

The program allows the user to specify a file name for the frame currently being constructed or altered, and then this file will be loaded onto the screen where alterations can be made in a very simple manner.

All Teletext/Mode 7 control codes can be entered and used on this editor, and the format of the screen is such that a frame saved from Prestel or Micronet is in the right form to be used with this editor. In fact, the original program allowed the user to alter the frame number and then upload the frame back on to Prestel, but as you need editing codes for this, the facility to upload has not been implemented here.

Firstly, then, an explanation of how to create and edit a frame. Having typed in the program and saved it, type RUN. As it is unlikely that you will have any previously saved frames unless you are a seasoned Micronteter, think of a filename and type it in. Press RETURN and you will then be asked if this is a new file or not. Pressing any key other than 'N' will be taken as meaning yes, so press one of these keys. The screen will clear, and the top and bottom lines will be set up to show that this is a new file and to display the current cursor position (H00 V00). Note also that 'Op' has appeared in yellow in the extreme top right hand corner of the screen. This is used by Prestel to represent the frame price, and is simply a convention.

Just below the blue cursor position display you will see a flashing cursor. Try using the arrow keys to move this about. Notice how the top line of the display changes to reflect the current cursor position, and that the cursor can only be moved within the area between the top and bottom lines – you cannot reach either of these lines. This is because a standard Prestel screen uses these two lines for system messages, so any user input here will be lost anyway.

Now move the cursor to the beginning of a line by pressing the RETURN key. RETURN only invokes a carriage return, NOT a line feed – Prestel conventions again! Type a few characters – anything you like. You will see them appear on the screen in the current cursor's position. Now press RETURN again, followed by ESCAPE and then capital 'M'. Everything that you have typed suddenly appears in double height. Now press ESCAPE followed by the closing square bracket next to the RETURN key. This creates a double height white background, but all the letters you typed beforehand have disappeared. Press ESCAPE followed by capital 'A', and they all re-appear in red. Easy, isn't it?

In the box below there is a full description of the entire Prestel character set and how to get them using this editor. Some of the control characters mentioned there, such as cursor on and cursor off, have no effect here as they are not relevant. Also, some control combinations are rather more powerful here than they are on Prestel. Here is a brief summary of the exceptions to the standard control set:

7 (CTRL-G or COPY) ends the edit and saves the frame.

17 (CTRL-Q) has no effect.

20 (CTRL-T) has no effect.

27 (CTRL-) behaves exactly as pressing the ESCAPE key.

138 (ESCAPE J) ends the edit without saving the frame.

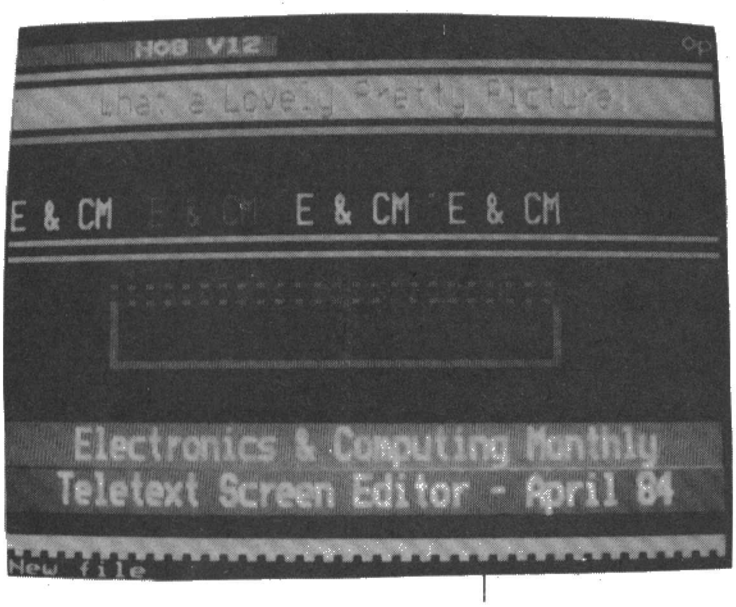
139 (ESCAPE K) puts character 139 on the screen but has no visible effect.

142 (ESCAPE N) moves all the text on the current line one position to the right from the current cursor, thus allowing a character to be inserted at the current position. The character at the end of the line is lost.

143 (ESCAPE O) deletes the character at the cursor and moves all the text on the current line following the cursor one position left. A space is inserted at the end of the line.

144 (ESCAPE P) produces a prompt at the bottom of the screen, asking whether you want to delete or insert a line. Pressing 'D' will delete the entire current line, moving all the lines below it up by one line and adding a blank line at the bottom, while pressing 'I' will insert a blank line into the current line and move all lines below it down by one line, losing the bottom line.

All cursor movements can be obtained in three ways: pressing the relevant arrow key, pressing CONTROL and the arrow key


HOB V12
What a Lovely Pretty Picture
E & CM E & CM E & CM E & CM
Electronics & Computing Monthly
Teletext Screen Editor - April 84
New file

The picture on the left took less than five minutes to construct. You should soon be able to do a good deal better in half the time!

together, or pressing CONTROL and one of the keys from H to K as indicated in the box below. The cursor can be homed and the screen can be cleared in the indicated ways, and pressing CTRL-X will delete all text on the current line from the cursor position to the end of the line without affecting the cursor position.

You may require to insert a linefeed or carriage return character on the screen in such a way that it occupies a position but does not affect the cursor. This can be done by pressing ESCAPE followed by CTRL-J, when you will be prompted for LF or CR. Pressing L will insert a linefeed character (CHR$ 10) while pressing C will insert a carriage return (CHR$ 13). Notice that a CR is treated like a double height character on the screen.

As can be seen, this editor is very comprehensive, and in fact there is one more facility that it offers: Prestel frames each have a page number of from 1 to 9 numbers followed by a lower case letter from a to z. By pressing CTRL-@ the top line of the display changes into a prompt for the frame number; up to 10 characters can be entered here, and DELETE operates in its normal way. This frame number will be saved along with the frame.

Finally, some notes about the listing. First and foremost, extensive use of direct screen access is made, so that this program will *not* work correctly with a second processor. Secondly, a number of the strings and PRINT statements used are actually coloured, but as this cannot be represented in the listing, there is a special convention that has been employed. Where there should be a colour control

Listing: Teletext graphics designer.

```
   10 REM (C) 1984 Adam Denning
   20 DIMZ% &398
   30 *FX229,1
   40 *FX4,1
   50 *TV255
   60 REPEAT MODE7
   70   ONERROROFF
   80   FORA%=0TO1:VDU141,157,129:PRINT"Teletext Frame Editor  (C) 1984 AD":NEXT
   90   PRINT'''"f6 Use the COPY key or CONTROL-G to exit"'
  100   PRINT"f6  Or ESCAPE J to exit without changes"
  110   INPUT''"f3Editing filename:f7"F$:IFLENF$=0 THENCLS:PRINT"f5Program Termi
nated":PROCend:UNTILTRUE:END
  120 PRINT''"f2Is this a new file (Y/N)?":S%=GET:IFS%>96 S%=S%-32
  130 MODE7:IFS%=ASC"N"THEN PROCreadfile ELSE $&7F98="f3New file":$&7C25="f30p"
  140 x%=0:y%=1:@%=2:n$="":PROCtopline:VDU28,0,23,39,0,31,0,1
  150 ONERRORGOTO160
  160 REPEAT c%=(GET AND&7F):IF c%=7 c%=-1:GOTO 220
  170   IFc%<>27 GOTO210
  180   c%=(GET AND&7F):IFc%=74 UNTILTRUE:UNTILFALSE
  190 IFc%>=65 ANDc%<=126 c%=c%+64
  200 IFc%<32 PROCcontout:GOTO220
  210 IFc%<>0 PROCoutchr ELSE PROCframe
  220 PROCadjust:UNTILc%=-1:PROCwritefile:UNTILFALSE
  230
  240 DEFPROCreadfile
  250 LOCALA$,A%,S%
  260 A$="f3Please wait - file being loaded":$&7F98=A$
  270 S%=OPENIN(F$):A%=0:REPEAT:?(Z%+A%)=BGET#S%:A%=A%+1:UNTILEOF#S%:CLOSE#S%
  280 FORA%=0TO&397:?(&7C00+A%)=?(Z%+A%):NEXTA%:FORA%=0TOLENA$:?(&7F98+A%)=0:NEX
T:ENDPROC
  290
  300 DEFPROCwritefile
  310 LOCALA$,A%,S%
  320 A$="f3Please Wait - File Being Saved":$&7F98=A$:FORA%=0TO&397:?(Z%+A%)=?(&
7C00+A%):NEXTA%:CLS:PRINTA$
  330 S%=OPENOUT(F$):FORA%=0TO&397:BPUT#S%,?(Z%+A%):NEXTA%:CLOSE#S%:ENDPROC
  340
  350 DEFFNdouble
  360 LOCALq%,doub,line%
  370 IFy%=22THEN=FALSE
  380 doub=FALSE:line%=40*y%
  390 FORq%=39TO0STEP-1
  400   IF?(&7C00+q%+line%)=141 THENdoub=TRUE
  410   NEXT
  420 =doub
  430
  440 DEFPROCduplic
  450 IFy%=22 THEN ENDPROC
  460 FORq%=0TO39:?(&7C00+y%*40+40+q%)=?(&7C00+y%*40+q%):NEXT:ENDPROC
  470
  480 DEFFNcurpos=x%+40*y%
  490
  500 DEFPROCdeleol
  510 LOCALxpos%,q%
  520 xpos%=39-x%
  530 IF FNdouble THEN FORq%=0TOxpos%:?(&7C00+FNcurpos+40+q%)=0:NEXT
  540 FORq%=0TOxpos%:?(&7C00+FNcurpos+q%)=0:NEXT:ENDPROC
  550
  560 DEFPROCinstchr
  570 LOCALq%
  580 IFx%=39 THEN ENDPROC
  590 IF FNdouble THEN FORq%=39-x%TO1STEP-1:?(&7C00+FNcurpos+40+q%)=?(&7C00+FNcu
rpos+39+q%):NEXT:?(&7C00+FNcurpos+40)=0
  600 FORq%=39-x%TO1STEP-1:?(&7C00+FNcurpos+q%)=?(&7C00+FNcurpos+q%-1):NEXT:?(&7
C00+FNcurpos)=0
  610 ENDPROC
  620
  630 DEFPROCdelchr
  640 LOCALq%
  650 IF FNdouble THEN FORq%=x%+41TO80:?(&7C00+y%*40+q%-1)=?(&7C00+y%*40+q%):NEX
T:?(&7C00+40*y%+79)=0
  660 FORq%=x%+1TO40:?(&7C00+y%*40+q%-1)=?(&7C00+y%*40+q%):NEXT:?(&7C00+40*y%+39
)=0:ENDPROC
  670
  680 DEFPROCcontout
  690 LOCALa$,q%
  700 a$="f6LF or CR? (press L or C)"
  710 IFc%<>10GOTO740
  720 $&7F98=a$:REPEAT c%=GET:IFc%>96 c%=c%-32
  730   UNTILc%=ASC"L" OR c%=ASC"C":FORq%=0TOLENa$:?(&7F98+q%)=0:NEXT:IF c%=ASC"
L" THEN c%=10 ELSE c%=13
  740 ?(&7C00+FNcurpos)=c%:PROCincx:ENDPROC
  750
```

```
  760 DEFPROCincx
  770 x%=x%+1:IFx%<>40 ENDPROC
  780 x%=0:y%=y%+1:IFy%=23 y%=1
  790 ENDPROC
  800
  810 DEFPROCnew_old
  820 LOCALa$,q%,c%
  830 a$="f6Delete or Insert line? (press D or I)"
  840 $&7F98=a$:REPEAT c%=GET:IFc%>96 c%=c%-32
  850   UNTILc%=ASC"D" OR c%=ASC"I":FORq%=0TOLENa$:?(&7F98+q%)=0:NEXT:IF c%=ASC"
D" THEN PROCtake ELSE PROCadd
  860 ENDPROC
  870
  880 DEFPROCtake
  890 LOCALq%,line%
  900 IFy%=22 THEN ENDPROC
  910 line%=y%*40
  920 FORq%=0TO879-line%:?(&7C00+line%+q%)=?(&7C00+line%+q%+40):NEXT
  930 FORq%=&7C00+880TO&7C00+919:?q%=0:NEXT
  940 ENDPROC
  950
  960 DEFPROCadd
  970 LOCALline%,q%
  980 IFy%=22 THEN ENDPROC
  990 line%=y%*40
 1000 FORq%=&7C00+919TO&7C00+line%STEP-1:?q%=?(q%-40):NEXT:FORq%=&7C00+line%TO&7
C00+line%+39:?q%=0:NEXT:ENDPROC
 1010
 1020 DEFPROCend
 1030 *FX229,0
 1040 *FX4,0
 1050 X%=0:Y%=0:A%=0:A%=USR(&FFCE)
 1060 ENDPROC
 1070
 1080 DEFPROCoutchr
 1090 IFc%=127 PROCdelete:ENDPROC
 1100 IFc%=24 PROCdeleol:ENDPROC
 1110 IFc%=142 PROCinstchr:ENDPROC
 1120 IFc%=143 PROCdelchr:ENDPROC
 1130 IFc%=144 PROCnew_old:ENDPROC
 1140 IFc%<32 THEN1180
 1150 c%=c%OR&80:VDUc%:IFFNdouble VDU31,x%,y%+1,c%
 1160 IFc%=141 PROCduplic
 1170 PROCincx:ENDPROC
 1180 IFc%=8 PROCdecx
 1190 IFc%=9 PROCincx
 1200 IFc%=10 y%=y%+1:IFy%=23 y%=1
 1210 IFc%=11 y%=y%-1:IFy%=0 y%=22
 1220 IFc%=12 x%=0:y%=1:VDU12
 1230 IFc%=13 x%=0
 1240 IFc%=30 x%=0:y%=1
 1250 ENDPROC
 1260
 1270 DEFPROCdecx
 1280 x%=x%-1:IFx%<>-1 ENDPROC
 1290 x%=39:y%=y%-1:IFy%=0 y%=22
 1300 ENDPROC
 1310
 1320 DEFPROCdelete
 1330 PROCdecx:IFFNdouble ?(&7C00+FNcurpos+40)=0
 1340 ?(&7C00+FNcurpos)=0:ENDPROC
 1350
 1360 DEFPROCframe
 1370 LOCALv$,q%,v%
 1380 $&7C00="f2FRAME NUMBER:f7":?&7C0F=0:VDU31,25,0:FORq%=0TO9:VDU32:NEXT:VDU31
,25,0
 1390 v$="":REPEATv%=GET:IFv%>31ANDv%<128 v$=v$+CHR$v%:IFv%=127THENv$=LEFT$(v$,L
ENv$-2)
 1400   IFv%>31ANDv%<128 THENVDUv%
 1410   UNTILv%<32ORLENv$=10:IFLENv$<10 THENFORq%=LENv$TO9:v$=v$+" ":NEXT
 1420 n$=v$+RIGHT$(n$,5):ENDPROC
 1430
 1440 DEFPROCtopline
 1450 LOCALq%,v%
 1460 FORv%=&7C19TO&7C27:n$=n$+CHR$?v%:?v%=0:NEXT:FORq%=&7C00TO&7C18:?q%=0:NEXT:
VDU31,0,0,132,157,135,32,32,32,32,32,72,48,48,32,86,48,48,32,156,31,25,0:PRINTn$
;:VDU31,x%,y%:ENDPROC
 1470
 1480 DEFPROCadjust
 1490 LOCALq%,h$,t$,v$
 1500 h$=STR$x%:v$=STR$(y%-1):IFx%<10 h$="0"+CHR$(x%+48)
 1510 IFy%<11 v$="0"+CHR$(y%+47)
 1520 t$="H"+h$+" V"+v$:VDU31,0,0,132,157,135,32,32,32,32,32:PRINTt$;:VDU32,156,
31,25,0:PRINTn$;:VDU31,x%,y%:ENDPROC
```

character, a lower case 'f' followed by a number has been placed. Instead of typing this combination in, press SHIFT and the function key indicated. Thus, if a line contains a yellow control character, this is represented by f3, as this is the function key that when pressed with SHIFT gives yellow.

Having edited or created one frame, the program then causes you to return to the start, asking for another file name. The only way of exiting from the program is to press RETURN at this point. The main program loop is controlled by lines 60 to 220, all the procedures being called from this section. This is where efficiency can be increased if you have a disc system, as the array Z% is not required. So all references to it can be removed and there is no need to ask the user if the file specified is a new one or not; this requires minor alteration to lines 120 and 130, and PROCreadfile.

Having obtained the filename and perhaps loaded it, a further REPEAT.. UNTIL loop is entered, starting at line 160. This reads the keyboard and reacts according to the key pressed. If the key was ESCAPE then the next key is read. Note that character 7, obtained by pressing CTRL-G or the COPY key, is treated as end of edit. ESCAPE J is also treated as end of edit, but does not cause the frame to be saved. The current cursor position is kept in the variables X% and Y%, and the frame number is kept in n$.

Now for a description of each of the procedures.

PROCreadfile: this opens the specified file and reads it into the array Z%. This array is then transferred to the screen, which starts at &7C00 and extends to &7F97, ignoring the bottom line.

PROCwritefile: this does exactly the obverse of the above, reading the screen into the array and then saving the array. Note that disc users could write a little bit of machine code to use OSGBPB instead of the interminable BPUTs and BGET.

*FNdouble:*this function returns TRUE if there is a double height character in the current line.

PROCduplic: duplicates the current line onto the line below, from the current position. This is used in handling double height characters.

FNcurpos: returns the offset from the start of the screen that represents the current cursor position.

PROCdeleol: deletes all text from the current cursor position to the end of the line.

PROCinstchr: inserts a space at the current cursor position, moving the rest of the line one space right beforehand.

PROCdelchr: does the opposite of instchr.

PROCcontout: allows for the insertion of control characters below ASCII 32 to be inserted onto the screen.

PROCincx: simply increments the current cursor position by 1.

PROCnewold: is called by pressing ESCAPE P and asks the user whether he wants to insert or delete a line, calling the appropriate procedures.

PROCtake: is called by newold, and deletes an entire line.

PROCadd: is also called by newold, and inserts a blank line.

PROCend: tidies up at the end of the program, closing any open files and setting the ESCAPE and cursor keys to give their default effects.

PROCoutchr: demonstrates how useful a CASE statement would be! It is full of IFs that are unnecessary in a higher level language, and simply produces the required effect of a key press.

PROCdecx: reduces the current cursor position by one.

PROCdelete: mimics the effect of the DELETE key, taking double height characters into consideration too.

PROCframe: allows the user to alter the frame number, and is called by pressing CTRL-@.

PROCtopline: creates the top line of the screen, including the cursor position indicator.

PROCadjust: sets the cursor position indicator and moves the cursor to that position.

For a relatively short program, the Editor is very powerful; and actually forms part of much larger multi-purpose Prestel terminal. A 6502 machine code version of this will be appearing on Micronet in the next few weeks, and details of this can be obtained by sending an SAE to the address below. The source code for this program, written in BCPL for the BBC Micro, can be obtained by sending £3.00 to the same address.

Adam Denning, c/o Micronet 800, Scriptor Court, 155 Farringdon Road, London EC1R 3AD.

Adam Denning is the Technical and Software Editor of Micronet 800.

Explaining the Prestel character set

Each Prestel character that you see on the screen (and even those you don't!) is represented in the computer by a number from 0 to 255. As far as the on-screen appearance goes, the characters from 128 to 255 are identical to those with codes 128 less. In other words, the character 'A' can be represented by the number 65 AND by the number that is 128 greater, that is, 193. Prestel itself sends and receives data using 7 bits, which means that no character sent is over 127. Nevertheless, this software displays the information on the screen using the full 8 bits, so that it can make the differentiation between control codes more easily.

Characters from 0 to 31 and from 128 to 159 are called 'control characters', because they control the layout of the screen. Thus, character 141 causes all characters on the rest of the line to be printed in double height, unless a character with the code 140 is met, when everything following this character returns to normal height. Naturally, double height characters take up two lines, and so the line below should be an exact duplication of the line above (including all the control characters) to achieve the desired effect.

Character 127 is also a control character, its effect being to move the cursor back one position and delete the character that is there.

Due to the sheer number of possible control characters, some have slightly different effects in the frame editor and the special characteristics of these keys are explained in the text for that program. All on-screen characters are the same as their online counterparts.

Possibly one of the hardest concepts to come to terms with in the character set is the difference between alphanumeric colour control codes (those from 129 to 135) and the graphic colour control codes (those from 145 to 151). These both have the same effect if one is typing capital letters, but lower case letters and other characters are interpreted somewhat differently if there is a graphic control code in front of them. Instead of the expected letters appearing, you will discover that block graphic characters are displayed, identical to the ones used in Teletext and Prestel pictures.

It is recommended that you spend some time experimenting with the frame editor, designing your own frames and observing the effects of each control character.

There now follows a brief summary of all the Prestel characters.:

```
0 - 7 these have no effect at all
8 (CTRL-H) cursor left
9 (CTRL-I) cursor right
10 (CTRL-J) cursor down
11 (CTRL-K) cursor up
12 (CTRL-L) cursor home and clear
screen
13 (CTRL-M) cursor return
14 - 16 treat as no effect
17 (CTRL-Q) cursor on in edit mode
18 - 19 no effect
20 (CTRL-T) cursor off in edit mode
21 - 23 no effect
24 (CTRL-X) delete to end of line
25 - 26 no effect
27 (CTRL-[) escape
28 - 29 treat as no effect
30 (CTRL-^) cursor home
31 - no effect
32 space
33 ! and graphics character 1
34 " and graphics character 2
35 £ and graphics character 3
36 $ and graphics character 4
37 % and graphics character 5
38 & and graphics character 6
39 ' and graphics character 7
```

```
40 ( and graphics character 8
41 ) and graphics character 9
42 * and graphics character 10
43 + and graphics character 11
44 , and graphics character 12
45 - and graphics character 13
46 . and graphics character 14
47 / and graphics character 15
48 - 57 numbers 0 - 9 and graphics chrs
58 : and graphic 26
59 ; and graphic 27
60 < and graphic 28
61 = and graphic 29
62 > and graphic 30
63 ? and graphic 31
64 @
65 - 90 Upper case A - Z

91 [
92 \
93 ]
94 ^
95 #
96 _ and graphic chr
97 - 122 lower case a - z and graphics
123 {
```

```
124 |
125 }
126 _
127 backspace and delete
128 nothing
129 alphanumeric red
130 alphanumeric green
131 alphanumeric yellow
132 alphanumeric blue
133 alphanumeric magenta
134 alphanumeric cyan
135 alphanumeric white
136 flash
137 steady
138 end edit in Prestel edit mode
139 start edit in Prestel edit mode
140 normal height
141 double height
142 start insert in Prestel edit mode
143 - 144 nothing
145 graphics red
146 graphics green
147 graphics yellow
148 graphics blue
149 graphics magenta
150 graphics cyan
```

```
151 graphics white
152 conceal
153 contiguous graphics
154 separated graphics
155 nothing
156 black background
157 new background
158 hold graphics
159 release graphics
160 - 255 a copy of the characters from
32 to 127
```

Further details can be obtained from a section of Prestel itself, *Understanding Prestel,* while the way to obtain these characters on the BBC Micro is to press either the key corresponding to the code you require, or in the case of codes that cannot be obtained this way, (that is, for codes from 128 to 159, press ESCAPE once, followed by the key that has a number equal to the value that you require *MINUS* 64. Thus ESC-M gives you double height (character 141) because M is represented by 77, and 77 + 64 is 141. These codes can only be accessed in this way from within software such as this, as pressing ESCAPE normally will produce a differing effect, depending on the language ROM currently in use. ■

ELECTRONICS & COMPUTING
MONTHLY
FREE FREE
GETTING YOUR WIRES CROSSED?
ORIC A D BOARD
THE ULTIMATE SPEECH BOARD
SINCLAIR QL - DEATH OF THE 8-BIT MICRO
PRINTSPOOLER SOFTWARE - PROGRAM WHILE YOU PRINT

Micrographics techniques

Advanced programming

Mike James extends the ideas of low-res user defined graphics and introduces the concepts of windowing and clipping.

Moving from creating and perhaps transforming single shapes in two dimensions to creating an entire screen display containing many objects is the most important problem in micrographics. The trouble stems from the simplicity of the idea of entering co-ordinates to define each shape and the sheer volume of work this creates even for a small number of objects. The direct approach of specifying co-ordinates for each object is simple only for the programmer – for the user it is a typical computer nightmare; masses of data that need to be entered to achieve anything at all. Obviously some method of generating standard shapes has to be introduced. This is similar to the situation in low-resolution graphics where user-defined graphics characters can be defined once and then re-used at different screen locations and in different colours as many times as required. In high-resolution graphics the equivalent of the user-defined character is the 'prototype'.

Prototypes and objects

In the previous two parts of micro graphics a point and line editor program has been used to illustrate the ideas that lie behind the storage of graphics data and transformations. In fact the point and line editor comes very low down in the hierarchy of modules that make up a full two-dimensional graphics package. The main use of a point and line editor would be to build up 'prototype shapes' that would form a library to be used in the construction of a complete display. Each prototype would be identified by a prototype number (or even a name) and this would be used to 'recall' the prototype from the library. The usual procedure is to add a newly recalled prototype to the display in a fixed position – the bottom right hand corner say – and then allow the user to move it and alter it so that it fits in with the whole display. For example adding a house to a landscape would involve recalling the house prototype and then altering its scale and position to fit in with the other houses already in the display. Notice that it is the range of ways in which a high-resolution graphics prototype can be changed that makes it different from a user-defined graphics character. In principle a prototype can be positioned (translated), scaled and rotated whereas a user-defined character can normally only be positioned on the screen. You should recognise that a prototype can be subject to any of the two-dimensional transformations described in last month's Micrographics.

The standard sequence of operations would be:-

1) use the point and line editor to draw any prototype shape that is needed.

2) recall each prototype that is needed and transform it to the correct position and orientation within the display.

3) finally use the point and line editor on the nearly completed display to draw in connecting lines etc. that are not part of any prototype. Also at this stage the editor can be used to remove unwanted lines and generally tidy up the final display.

Unfortunately this is the point at which the size of the programming project involved makes it impossible to include a BASIC example of using object prototypes. The only convincing demonstration that such a system works is to produce a complete graphics package. Even though this would take far too much room it is possible to give a description of the additions to the data structures etc. that are needed to develop the simple point and line editor into such a package.

Graphics package

The point and line editor developed over the past two months is easy to convert to a prototype editor. All that is needed is an additional array, P, to hold a prototype number. Thus the co-ordinates of each point in the entire prototype library are

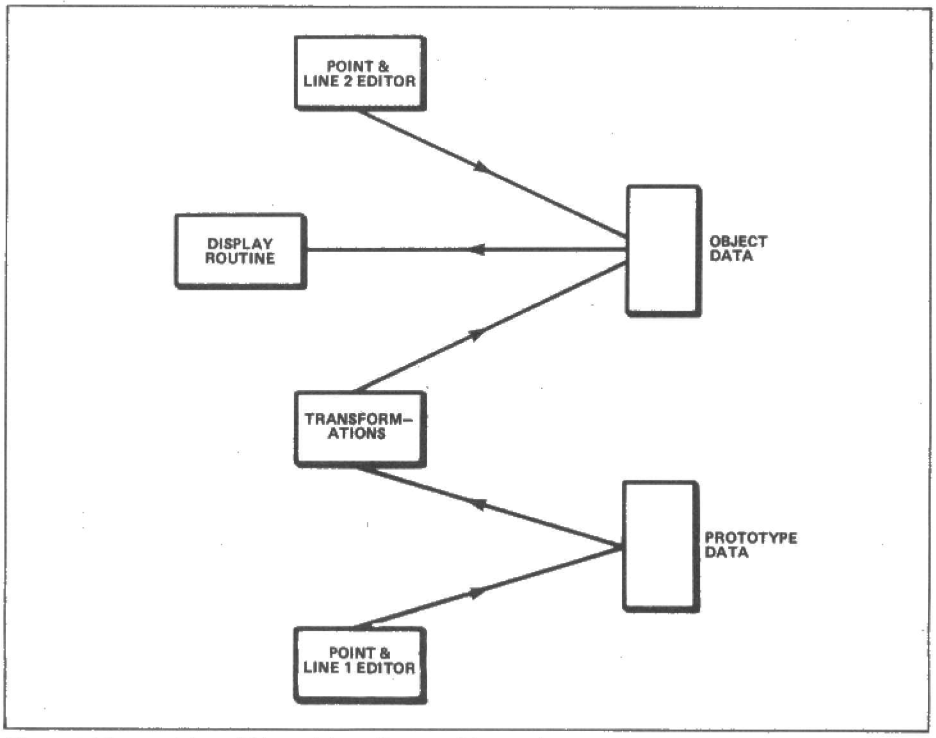


Figure 1. The modules within an interactive graphics package.

stored in the'arrays X and Y. Each prototype is identified by a number and this number is stored in P to identify which lines in the line file constitute the prototype. In other words the line X(S(I)),Y(S(I)) to X(E(I)),Y(E(I)) belongs to prototype P(I). (To follow this you have to know that S(I) holds the index of the point in the point file that starts the Ith line and E(I) holds the index of the end of the Ith line).

So to define a new prototype all you have to do is store points and lines along with the prototypes number. To display prototype PR all that you have to do is (in a cross between English and BASIC)

```
FOR I=1 TO number of prototypes
IF P(I)=PR THEN draw line I
NEXT I
```

In other words, draw all the lines that correspond to the prototype. In the same way the prototype can be transformed by applying the desired transformation to all the points that form the start and end of the lines of the prototype. When the final transformation that places the prototype at the right position, at the right size and orientation has been determined, it can be applied to the points of the prototype and the result transferred to separate point and line files that are used to store the actual objects that make up the display. These additional point and line files have the same structure as the ones used to hold the prototypes and can be operated on by a similar point and line editor, and even a transformation package, to 'clean up' and 'tinker with' the final display.

A finished interactive graphics package is no small program and needs careful planning but as you can see from **Figure 1** a number of the modules are very similar to one another and this helps to reduce the complexity.

Windowing

So far it has been assumed that the desired two-dimensional image is small enough to fit on the screen. However in practice this is rarely the case. For example, most electronic circuit diagrams are too large to display in their entirety on a standard screen. To a certain extent this doesn't matter in that the engineer or anyone examining the diagram is most likely to want to view only a part of it at any one time. The problem of displaying only a small part of an entire image is called 'windowing' and surprisingly it is related to the subject of transformations dealt with last month.

'windowing solves the problem of displaying only a small part of the image on screen'.

The plan of the interactive graphics package given earlier must now be expanded to include the possibility that the co-ordinates stored in the point file cover a larger range than the screen can accommodate. The co-ordinates within the display file are often referred to as 'world' co-ordinates and the problem is to allow the user to specify a rectangular 'viewing window' in the world co-ordinates and then find a transformation that will 'map' everything in the window to the range of co-ordinates available on the screen. This transformation is known as a 'viewing' transformation and is even more important in three-dimensional graphics than in two. While applying the viewing transformation to all the points in the point file it is worth incorporating one extra feature. As well as not viewing all of the world co-ordinates in one go, it is also possible that you might not want to use all of the screen at once. For example, you might want to use only the top left hand quarter for graphics and the rest of the screen for presenting other information. The portion of the screen that you want to use to draw the lines within the viewing window is called a 'view port'. Now the viewing transformation has to transform all of the world co-ordinates within the viewing window into the screen co-ordinates within the view port, see **Figure 2.**

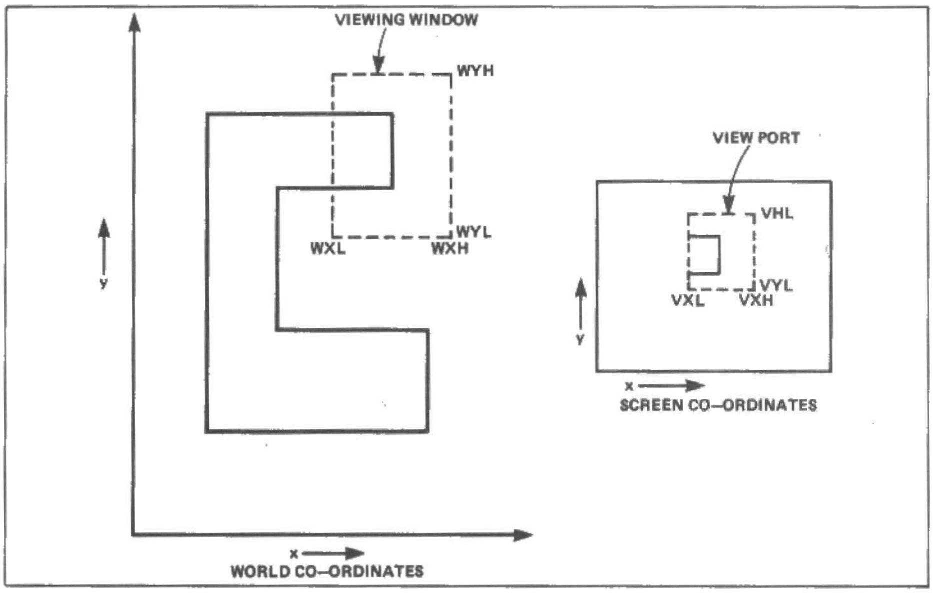


Figure 2. The viewing transformation.

It comes as a pleasant surprise to discover that the viewing transformation is not at all complicated. If the viewing window is a rectangle with edges at WXL, WXH, WYL and WYH (all measured in world co-ordinates) and the viewing port is a rectangle with edges at VXL, VXH, VYL and VYH (all measured in screen co-ordinates) as shown in **Figure 2,** then the transformation from world co-ordinates to screen co-ordinates is:-

xs = (VXH—VXL)*(xw—WXL)/(WXH—WXL) + VXL
ys = (VYH—VYL)*(yw—WYL)/(WYH—WYL) + VYL

where the point in the world co-ordinate system is xw,yw and the corresponding point in the screen co-ordinate system is xs,ys.

'the clipping process is a little more complicated than you might expect'.

Clipping

If you simply use the viewing transformation as it stands then every point within the viewing window will be transformed to a point within the view port but points just outside the window will also be transformed to points on the screen not in the view port. What is needed is some way of restricting the drawing of graphics information to the rectangle formed by the view port. This process is known as 'clipping' and is a little more complicated than you might expect. If you have access to the machine code routines that draw lines then all you have to do is not to plot any point that falls outside the viewing port. In most cases however the machine code routines are hidden and unchangeable inside a

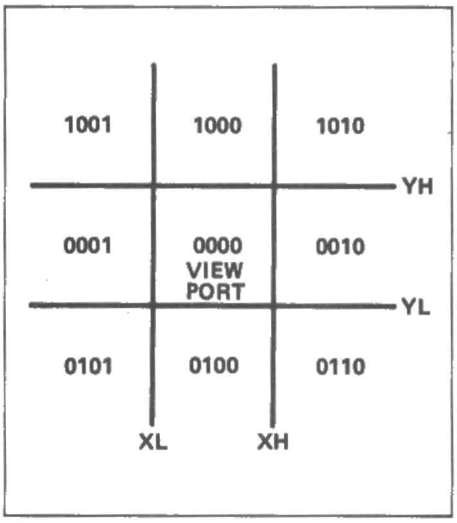


Figure 3. Screen division clipping.

ROM so the problem comes down to simply drawing the parts of lines that fall within the viewing port.

One of the best and simplest line clipping methods is based on the way that the edges of the viewing port divide the screen co-ordinate system into nine regions – see **Figure 3.** Each region is identified by a four bit code that is constructed in such a way that it is easy to work out the code of the region in which a point falls from its co-ordinates. The algorithm to construct the code for a point x,y is:-

set b0 if the point is to the left of the left hand edge of the view port
set b1 if the point is to the right of the right hand edge of the view port
set b2 if the point is below the bottom edge of the view port
set b3 if the point is above the top edge of the view port

Using this algorithm it is possible to discover the region code for the two end points of a line and it is obvious that the whole of the line is visible if both codes are zero (because this implies that both ends of the line are within the view port). What is less obvious is that if the bitwise AND of the two codes is not zero then no part of the line crosses the view port and therefore the entire line is invisible (ie should not be drawn)!

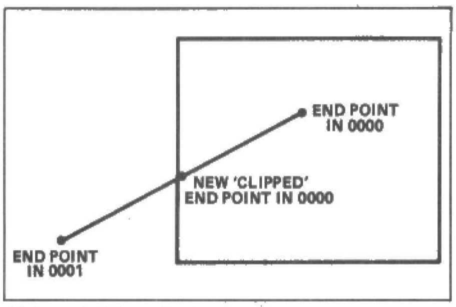


Figure 4. Edge clipping.

Thus, using this simple coding of the regions we can immediately detect those lines that are entirely within the view port and those that are entirely outside it. All the other lines must cross one of the view ports edges and consist of at least one part that is invisible and one part that is visible. To remove or clip the invisible portion of the line all that has to be done is to move the end point that is outside the view port 'down the line' to the point that it crosses the edge, see **Figure 4.** Once again the code can be used to discover which of the end points are outside the view port and which edge the line crosses. Moving one of the end points to the position where the line crosses the edge is just a matter of simple co-ordinate geometry. The only complication is that a line may cross more than one edge and hence may need to be clipped more than once. This is best achieved by simply repeating the whole clipping process over again with one of the endpoints changed to lie on the boundary.

A BASIC clipper

Program 1 implements the clipping algorithm described above. It is written in BBC BASIC but as none of that machine's special facilities are used it should be easy to convert it to other machines. The co-ordinates of the view port are set up by subroutine 9000 in XL, XH, YL and YH and the program then draws lines at random between X1,Y1 and X2,Y2. To see this without clipping change line 40 to:-

```
40  V=1
```

line 50 uses V to determine if a line is visible (V=1) or invisible (V=0) so setting V=1 makes all the random lines visible. With line 40 restored to its original state, subroutine 3000 is called which both sets V to indicate whether or not the line is visible and also clips lines to fit within the view port.

The subroutine structure of **Program 1** is shown in **Table 1.**

Table 1

line number	action
10–60	main program
1000	generate pairs of random endpoints in X1,Y1 and X2,Y2
2000	draw line between X1,Y1 and X2,Y2
3000	clip lines until both endpoints are within the view port
4000	calculate region codes for both end points return as C1 and C2
9000	set co-ordinates of view port and draw its edges

Subroutines 3000 and 4000 can be used in other programs to ensure that lines are confined to a specified view port.

Conclusion

This month's Micrographics brings the subject of two-dimensional graphics to a close. Two dimensional graphics does form the foundations upon which three-dimensional graphics is built so the time spent in understanding data presentation, transformations and the viewing transformation in particular will make it possible to introduce three-dimensional methods very quickly next month. However, it is important to realise that Micrographics has hardly scratched the surface of two-dimensional graphics and there are many specialised techniques that have been ignored. Most of these can be worked out from first principles as long as you understand what is required by the application and the fundamentals that have been covered. There is no denying that three-dimensional graphics produce results that are stunning but most applications need nothing more complicated than the techniques appropriate to two dimensions.

Next Month – Three dimensional graphics.

Program 1

```
  10 MODE 4
  20 GOSUB 9000
  30 GOSUB 1000
  40 GOSUB 3000
  50 IF V=1 THEN GOSUB 2000
  60 GOTO 30

1000 X1=RND(1279)
1010 Y1=RND(1023)
1020 X2=RND(1279)
1030 Y2=RND(1023)
1040 RETURN

2000 MOVE X1,Y1
2010 DRAW X2,Y2
2020 RETURN

3000 C1=0
3010 C2=0
3020 GOSUB 4000
3030 IF C1=0 AND C2=0 THEN V=1:RETURN
3040 IF (C1 AND C2)<>0 THEN V=0:RETURN
3050 C=C1
3060 IF C1=0 THEN C=C2
3070 IF (C AND 1)=1 THEN
          Y=Y1+(Y2-Y1)*(XL-X1)/(X2-X1):X=XL
3080 IF (C AND 2)=2 THEN Y=Y1+(Y2-Y1)*(XH-X1)/(X2-X1):X=XH
3090 IF (C AND 4)=4 THEN Y=YL:X=X1+(X2-X1)*(YL-Y1)/(Y2-Y1)
3100 IF (C AND 8)=8 THEN Y=YH:X=X1+(X2-X1)*(YH-Y1)/(Y2-Y1)
3110 IF C=C1 THEN X1=X:Y1=Y
3120 IF C=C2 THEN X2=X:Y2=Y
3130 GOTO 3000

4000 IF X1<XL THEN C1=1
4010 IF X2<XL THEN C2=1
4020 IF X1>XH THEN C1=C1 OR 2
4030 IF X2>XH THEN C2=C2 OR 2
4040 IF Y1<YL THEN C1=C1 OR 4
4050 IF Y2<YL THEN C2=C2 OR 4
4060 IF Y1>YH THEN C1=C1 OR 8
4070 IF Y2>YH THEN C2=C2 OR 8
4080 RETURN

9000 XL=300
9010 XH=800
9020 YL=300
9030 YH=600
9040 MOVE XL,YL
9050 DRAW XL,YH
9060 DRAW XH,YH
9070 DRAW XH,YL
9080 DRAW XL,YL
9090 RETURN
```

SINCLAIR SECTOR

Sinclair Sector is a new E&CM regular to keep you up to date with the latest hardware and software developments for ZX Spectrums and ZX81s. Compiled by Stephen Adams.

To start with the greatest show for Sinclair users, the ZX Microfair, achieved a new record attendance of over 12,000 people. Thanks mainly I think to a BBC News programme called "60 minutes" which at last thought Sinclair computing was respectable enough to get a mention. At the fair over 250 stalls were competing for customers and most were showing new products. The next ZX Microfair will be on the 28th April 1984 at Alexandra Palace, London. I suggest you get an advance ticket as soon as possible to avoid the queue (you just walk straight in).

Storm software and Widgit software were showing their educational games to teach the 5-7s how to recognise shapes and draw with their Spectrums.

New games were available from various sources and most were demonstrated using the "Quickshot 2" joystick which seems to be becoming quite popular.

Kempston were showing a new Centronics printer interface (the Model E) which contains software in EPROM allowing you not only to print, but also to COPY dot by dot the ZX Spectrum screen onto various printers including the EPSONs, NECs and Seikoshas. The EPROM sets up the new printer commands on powering up, so there is no software to load and as it exists in the ROM space no user RAM is needed.

Two disc systems were on view, Technology Research were showing one for the machine code enthusiast which allows you to use double sided discs as one disc giving 194K on 40 track discs or 388K on 80 track discs. Up to two disc drives may be attached to the one interface. Cost is £89.

Interactive Instruments were showing their BASIC disc system, which gives over 100K on a 40 track single sided disc drive for £228.85 (drive included). The controls are via BASIC variables which call machine code routines in the top 8K of memory. These allow you to SAVE and LOAD programs or data arrays from within a program. More information on these two systems will appear in a review next month.

The business software firm Transform have already converted their software to run on Interactive discs and they will soon support 80 tracks and double sided, twin discs. The software is at present being sold under the "Vicount" label by the Spectrum group of retail shops.

Sinclair Research had the QL computer there, but only the keyboard in it's case. The review models for various computer magazines still had not arrived by the end of February, but book shops were reported to have sold out of 68000 books as everyone was swatting up on how to write software on the new micro. The QL is based on a 8-bit version of the 68000 microcomputer chip called the 68008 and has a standard Eurocard interface.

This computer seems to have a great future, but most people I have spoken with would prefer a disc drive attached (rather than relying on the Microdrives as the only way of storing data while the computer is switched off). There is no cassette interface on the QL and any delays by Sinclair in producing units for the trade will push back any possibility of this, as Sinclair has no intention of producing one himself, only a hard disc interface at some later date.

Send any ZX news to
Stephen Adams c/o E&CM
155 Farringdon Road,
London EC1R 3AD.

Useful addresses

Technology Research Ltd., 356 Westmount Road, London SE9 1NW. Tel: 01-856-8408. Telex: 896691 TLXIR G.

Interactive Instruments, Unit 6, Pilot House, King Street, Leicester. Tel: 0533-551594.

Richard Shepard Software, Elm House, 23 Elmshott Lane, Cippenham, Slough, Berks. Tel: 06286-63531.

Kempston Microelectronics, Unit 30 Singer Way, Woban Road Industrial Estate, Kempston, Bedford. Tel: 0234-856633. Telex: 826078 KEMPMI G.

Transform Ltd., 41 Keats House, Porchester Mead, Beckenham, Kent. Tel: 01-658-6350.

Widgit Software, 48 Durham Road, London N2 9DT. Tel: 01-444-5285.

Storm Software, Winchester House, Sherborne, Dorset. Tel: 0935-813528.

Dream Software, PO Box 64, Basingstoke, Hants RG21 2LB. Tel: 0256-25107.

David Husband, 2 Gorleston Road, Brabksome, Poole BH12 1NW. Tel: 0202-302385.

Cambridge Microelectronics Ltd., 1 Milton Road, Cambridge CB4 1UY. Tel: 0223-31414 Telex: 81574.

Campbell Systems, 15 Rous Road, Buckhurst Hill, Essex IG9 6BL. Tel: 01-504-0589.

CPS, Shire Hall, Appleby, Cumbria CA16 6XN.

ZX Microfair, 71 Park Lane, London N17 0HG. Tel: 01-801-9172.

Meanwhile Classified Products and Services (CPS) have produced a flexible ribbon cable with snap-on IDC edge connectors for £8.50. The maximum length for a ZX Spectrum is about 3 inches without some sort of buffering, but you can fit several edge connectors to this length. This should solve the perennial problem of connecting two or three "dead ended devices" like printer interfaces to the Spectrum or ZX81.

Microdrives so I hear are due to hit the shops very soon, but there is still a lack of software being converted for them. Campbell's MASTERFILE is one such product and it has been enhanced to give up to 51 characters per line! Ideal for storing indexes to magazines such as this one. The Cash Controller by Richard Shepard is the only other program that boasts Microdrive compatibility.

For those who like to program their best programs into EPROMs, Cambridge Computing now produce an EPROM loader as well as programmer. This will load software on powering up the Spectrum from an 2K-8K type EPROM running them automatically. Ideal for control work or demonstrations as well as your favourite games.

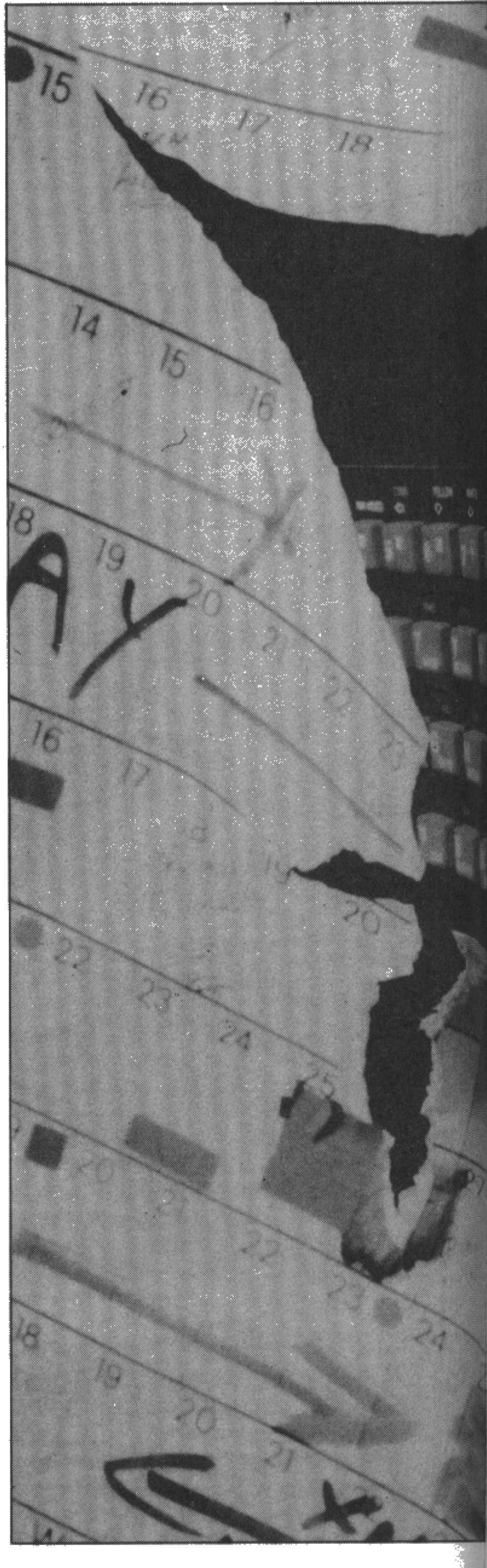
For more serious users Dream software have produced a CAD package which allows you to produce high quality graphics displays on the Spectrum. Unfortunately you cannot annimate them, but they do include 3 by 3 character "sprites" which can be placed anywhere on the screen. The cost is also reasonable at £4.95.

ZX81s seems to be catching on as control systems in industry and defence. The Ministry of Defence is apparently buying David Husbands multitasking FORTH boards to fit to ZX81s (but they don't say what they are doing with them). This FORTH is also one of the few programs that gives "windows" like the APPLE Lisa system. A ZX Spectrum version will soon be available, with bank switching, to allow you to switch backwards and forwards from BASIC to FORTH and will be compatible with the ZX Microdrive.

Hopefully this will become a regular item in *E&CM* but if you know of some device which you think is interesting let us know.

Spectrum real-ti

The Spectrum's internal timer is nothing but a useless piece of plastic and silicon when the power is switched off: it is therefore impossible to commit those irritating but important details of life to the computer. Richard Sargent's answer is a real-time clock with battery back-up, complete with an electronic diary in software.



Most computers these days have internal timers which can be configured to count and display in hours, minutes and seconds, but they cannot tell you what day of the week it is and whether or not your library books are overdue! This low cost project describes a real-time card which has battery back-up and plugs into the Spectrum expansion port. Its applications include an electronic diary, document dating and expiry-date checking in business, and long-period timing in science.

Choosing the chip

There are a number of low-power CMOS timer chips on the market, and although they are still rather expensive at the moment their prices are following the predictable downward trend. Some of the timer chips also have a certain amount of scratch-RAM built into them and although this is enough to store a few alarm-time values it isn't sufficient for the storage of multiple dates and events. This shortcoming is rectified by having a second CMOS chip of static RAM to provide the diary function of the real-time clock.

The first chip considered was the Mostek MK3805. Its specification looks good: CMOS technology, automatic leap-year correction, 24 or 12 hour format, 24 bytes of scratchpad RAM. Unfortunately it communicates with the host CPU via a serial data stream and the quoted price is £18.53. Goodbye MK3805. The next contender was the Hitachi HD146818, a cheaper chip with more facilities. In addition to the clock and calendar functions, it has 50 bytes of scratch-RAM and three types of interrupt output. The drawback with this chip is that although it has an 8-bit bus the pins are multiplexed with address lines 0-7 and the timing signals are those of the HD6801/HD6301 CPUs. By a process of elimination we are left with the MM58174 which is not only suitable, but won't strain the bank account too much either. It has the proper timekeeping facilities – 24hour clock and leap-year calendar. There is no on-chip memory, but interrupts are available. **Figure 1** shows the pin designation of this timer chip. The CMOS RAM presents no problems since the industry standard 6116 2K x 8 chip fits the bill perfectly.

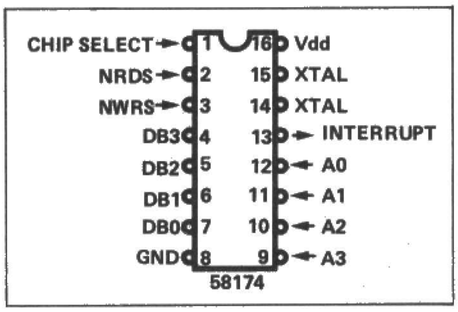
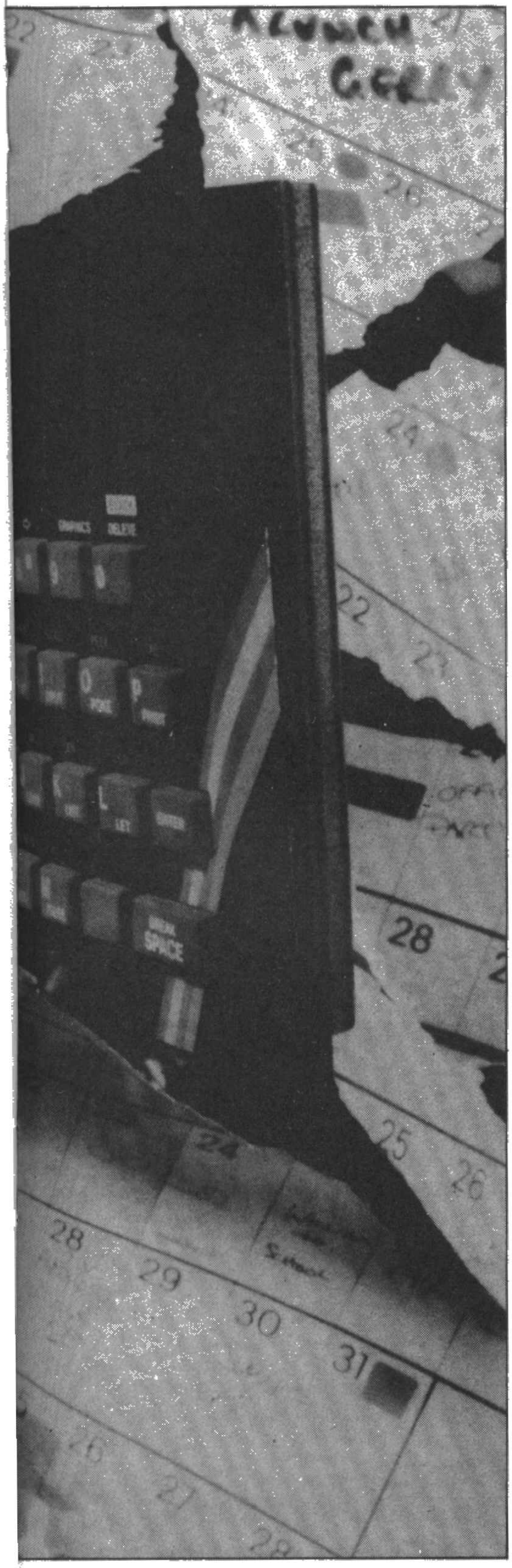


Figure 1. Timer chip pin designation.

Controlling the MM58174

When the chip is first powered up it is necessary to enter the correct data into the device registers and start the clock running. The low nibble of the data bus is used to pass BCD data to a correctly addressed register and in this manner all the internal counters are set to the desired time. (See **Table 1**). When the time-keeping registers have all been set, the clock is started by sending a high on DB0 to register 14. Con-

me clock



versely a low written there will stop the clock counting. Incidentally, all starts reset the seconds counter to zero. Don't forget to write data 8 to register 15 if you're building this card in 1984 (see **Table 2,** the leap year status register). By some strange oversight there is no readout capability on this register, so the micro can't tell if the current year is a leap year! Obviously one of the first pieces of information that will go into the CMOS RAM is the current year and a leap year status flag.

Reading data from the other registers is simply a matter of giving an IN command: the high nibble data is always read as zero due to the presence of IC5 and the low nibble is a valid clock or calendar value. There is one exception to this. If a register is updated during a read operation the I/O data is prevented from updating and a subsequent read will return the illegal BCD code 1111. This enables the detection of the fact that the previous data had changed and is now incorrect. This poses some problems to software running under BASIC as any program running slowly will tend to pick up quite a few "1111" codes, so the time-reading code takes up quite a few lines of BASIC.

Register 15 can be programmed as an interrupt timer giving 0.5, 5.0 or 60 second intervals and can be coded for single or repeated operations. The open drain interrupt output is pulled to ground when the timer times out and reading the interrupt register provides status and internal selected information. See **Table 3.**

Putting the clock on the map

Finding just over 2K of spare memory space is often rather difficult on the more modern home computers, and the Spectrum is a good case in point. However, since both the clock and the memory are simply data-holding devices, they can be mapped onto the Spectrum's I/O map and users of other Z80 CPU systems can take this solution too. There is also a second and less obvious advantage in this method. On a Z80 Input or Output operation a single "wait state" is automatically inserted by the CPU in recognition that MOS and CMOS I/O devices have difficulty in running at full CPU speed. For the Spectrum computer the RD/WR/IORQ pulses for I/O operations are all about 725 nanoseconds duration and this allows slow

No	Name	Mode	Address
0	Not used		
1	Tenths of sec	R	F19F 61855
2	Units of sec	R	F29F 62111
3	Tens of sec	R	F39F 62367
4	Units of mins	R/W	F49F 62623
5	Tens of mins	R/W	F59F 62879
6	Units of hours	R/W	F69F 63135
7	Tens of hours	R/W	F79F 63391
8	Units of days	R/W	F89F 63647
9	Tens of days	R/W	F99F 63903
10	Day of week	R/W	FA9F 64159
11	Units of month	R/W	FB9F 64415
12	Tens of month	R/W	FC9F 64671
13	Years	W	FD9F 64927
14	Stop/Start	W	FE9F 65183
15	Interrupt	R/W	FF9F 65439

The xx9F element in the address represents

	A7	A6	A5	A4	A3	A2	A1	A0
the bit pattern	1	0	0	1	1	1	1	1

which decodes as port E at the 74138 outputs.

Table 1. Internal Registers.

	DB3	DB2	DB1	DB0
Leap year	1	0	0	0
Leap year + 1	0	1	0	0
Leap year + 2	0	0	1	0
Leap year + 3	0	0	0	1

The register is a shift register and the contents are rotated to the right every 31st December.

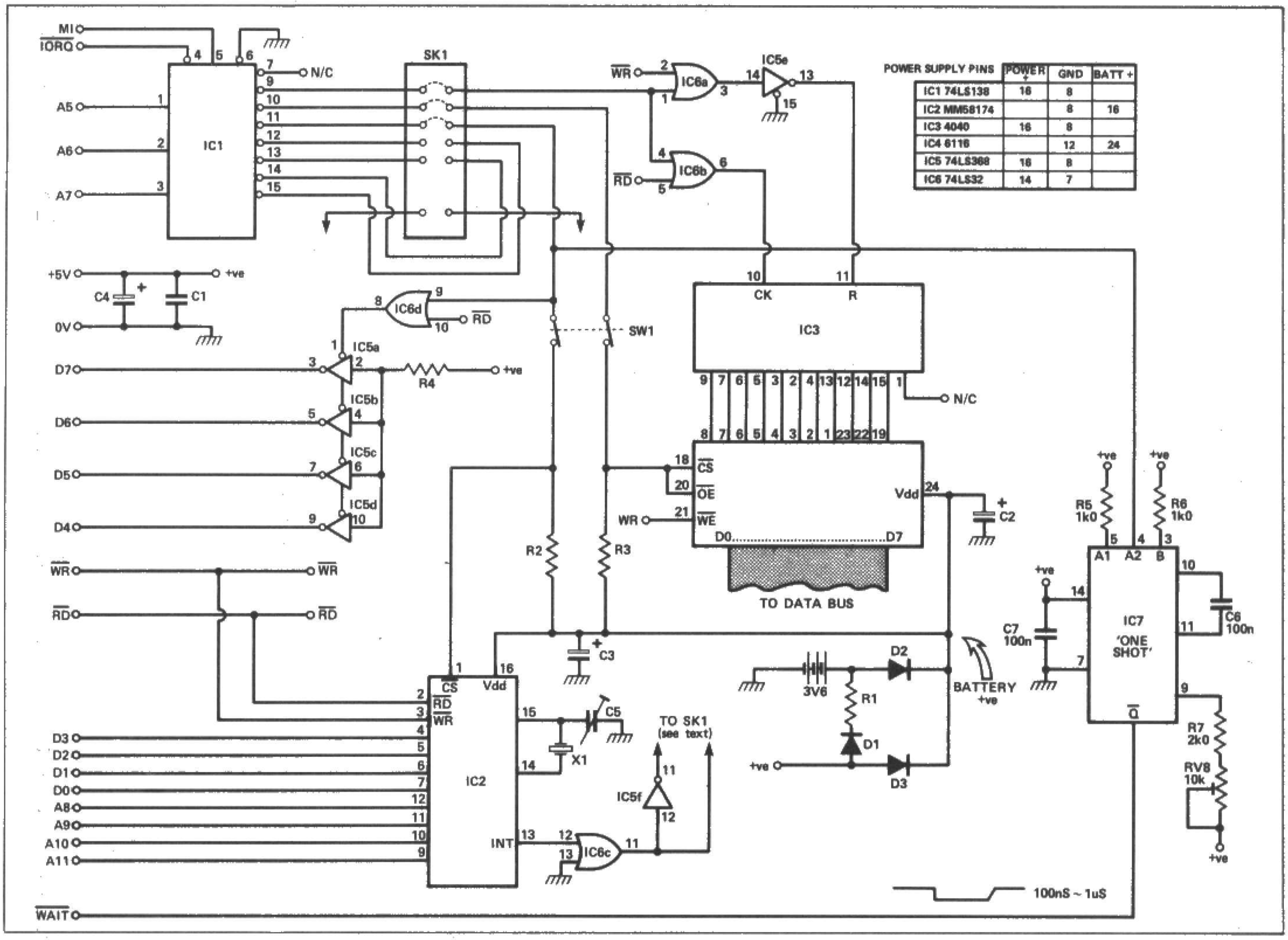
Table 2. Year Status Register

Function	DB3	DB2	DB1	DB0
No interrupt	0	0	0	0
Interrupt at 60sec intervals	0/1	1	0	0
Interrupt at 5.0s intervals	0/1	0	1	0
Interrupt at 0.5s intervals	0/1	0	0	1

Write mode:
DB3=0 single interrupt DB3=1 repeated interrupt

Read mode:
DB3=0 no interrupt DB3=1 interrupt given

Table 3. Interrupt Read/Write Register



POWER SUPPLY PINS	POWER +	GND	BATT +
IC1 74LS138	16	8	
IC2 MM58174		8	16
IC3 4040	16	8	
IC4 6116		12	24
IC5 74LS368	16	8	
IC6 74LS32	14	7	

Figure 2. Circuit diagram.

peripherals some time to get their data on to the bus (in the case of a CPU read operation) and a hardware implementation of the WAIT state should be unnecessary. However, the Spectrum runs a fast CPU (3.5MHz) and the *worst-case* specification for the delay between the MM58174 accepting a valid address and putting out valid data is given as 1200 nS. Fortunately most ICs are very much better than their worst-case specification, but as a safeguard again the possibility of "slow" MM58174 devices, IC7 will generate a WAIT pulse which will "stretch" the Spectrum's own RD/WR/IORQ pulse. With C6 at 100pF and R7 and VR8 together equalling 10K, IC7 will generate a low going pulse of 775nS which will lengthen the CPU's operations sufficiently to ensure that valid data is collected from the MM58174.

Circuit description

The port decoder IC1 takes the 3 address lines available for the user (A5,A6,A7) and enables them at IORQ time into seven user ports A-G. Port H is activated by Sinclair I/O operations and must not be used. Three of these ports are needed, the SK1 is provided on the PCB for the user to select which ports control the real-time card, since there may well be other add-ons using this area of the port map. A7 is used by 48K Spectrums, so ports A,B,C & D are avoided and the software assumes that ports E, F and G are used. Port E enables the timer chip, but as the chip has 16 internal registers they must be addressed by A8-A11. The address values of these registers are given in **Table 1.** Notice that the values are calculated by the bit-state of A5-A11, assuming that A0-A4 and A12-A15 are in a high state. (In fact the state of A0-A4/A12-A15 is irrelevant). The decimal values may be used directly with Spectrum IN and OUT commands to interrogate and alter the clock. The oscillator is formed by an on-chip inverter/amplifier with bias resistors and capacitors. A standard 32.768 kHz crystal is needed across pins 14 and 15 and it is recommended that a 5-65pF trimmer be used to fine tune the oscillator. Vdd is connected to the battery supply line and the chip will maintain its data in standby mode on voltages down to 2.2v.

The memory chip (IC4) is enabled by port F, when one byte may be written to or read from. The byte which is accessed is determined by the count on IC3, the CMOS 4040, which has its output pins connected directly to the address lines of IC4. A dummy write to port G resets the counter, while a dummy read to port G increments the counter. Thus the memory chip can be accessed in serial fashion, one byte at a time. IC3 needs a low-going pulse to increment its counter, and this is supplied, but a high-going pulse is needed to reset it. Thus an inverter precedes pin 11 of IC3. The spare gates of IC6 and IC5 are used to buffer and complement the output of the interrupt pulse which is available on pin 13 of IC2.

The Spectrum 5 volt rail is capable of supplying the 1mA needed to trickle charge the small NiCad battery which provides a nominal 3.6V standby voltage to IC2 and 4. The power-down arrangements are provided by SW1A/B and are semi-automatic. When the Spectrum is switched off the outputs of IC1 are seen as "high" by ICs 2 and 4 which remain on power, taking their supply from the battery through diode D1. ICs 2 and 4 are de-selected, with their data bus's high impedance. However there is always the chance that spurious pulses may come through on the WR line and on a CE line at the instant of switch-off, in which case a byte of data in the timer or the memory would be corrupted. As a safeguard against this happening at power off and power on times it is recommended that SW1 be opened (standby mode) *prior* to switch-off and closed (on mode) *after* switch-on.

Next Month: PCB foil pattern and software.

The ultimate speech synthesiser

In part 2 of this project, Russell Vowles* describes the construction and testing of the synthesiser and provides a list of commonly used words together with a breakdown of their allophone components.

***LB Electronics**

Construction should be straightforward especially if the ready made PCB is used to mount the components. The artwork for the PCB is shown in **Figure 1** and the component layout for the topside of the PCB is shown in **Figure 2.**

IC3 is manufactured using MOS circuit techniques and should therefore be handled with care, ie, leave it in its protective packaging until it is time to fit it onto the board and do not touch the pins as static charge could damage the device.

We recommend that the components should be soldered to the board with a miniature soldering iron and in the following order:-

1. Wire links and Vero pins.
2. Q1.
3. Capacitors (not C14).
4. Resistors and presets.
5. IC sockets.
6. Cable.
7. C14.

The cable should be prepared as in **Figure 3** for the Dragon 32 and as in **Figure 4** for the BBC Model A or B. The BBC machines do not have a 5V connection on the centronics port and so this is taken via the DC power connection at the rear of the BBC's case.

The connectors for the centronics interface require no soldering and are fitted by squeezing the cable between two halves of the connector in a small vice. This must be done carefully if the connector is not to be damaged.

After the connector has been fitted the cable may then be soldered to the Vero pins. A small piece of sleeving should be placed over each wire to prevent any short circuiting.

The loud speaker is connected using two flying leads which are soldered to the two speaker pins on the PCB. The ICs can now be inserted into their sockets, checking for correct orientation.

Testing

1. Switch off computer and plug speech synthesiser into printer port.
2. Adjust RV1, RV2 and RV3 to mid-way position.
3. Switch on computer and type in **Program 1** (for the Dragon 32) or **Program 2** (for the BBC).
4. Run the program and if all is well the computer should be heard to say "Hello" in four different pitches.

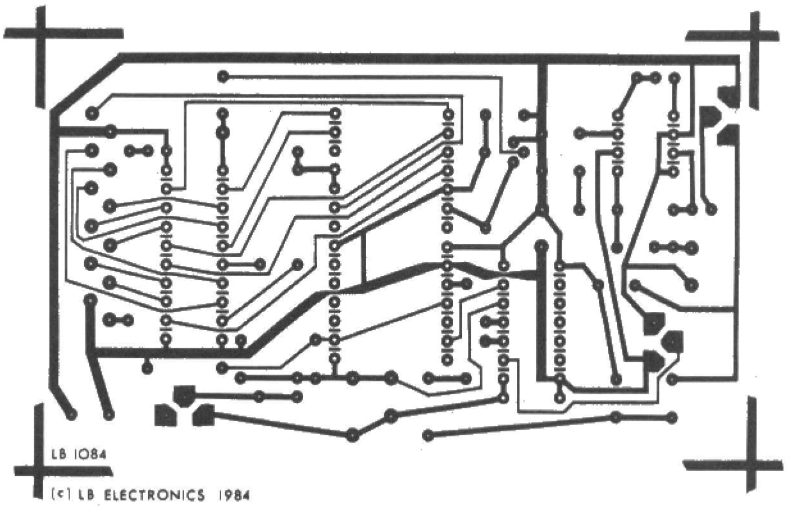


Figure 1. PCB foil pattern.

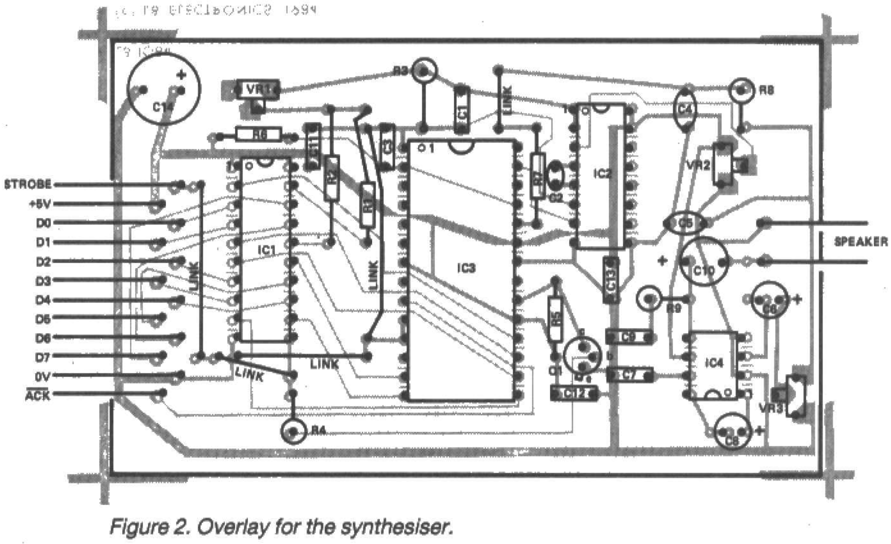


Figure 2. Overlay for the synthesiser.

5. If the pitches are too high or too low RV2 should be adjusted for correct pitch.
6. If the speech is too loud and distorted or it is too quiet, adjust RV3.
7. Adjust RV1 for the desired amount of inflection.

Fitting into the box

The box should be drilled as in our photographs (a ready drilled box is available for those less confident) and a small slot filed to one side of the box and on the lid to allow a passage for the cable.

The speaker can then be fixed over the holes in the box using Epoxy resin or sticky pads. The cable is fixed to the side of the box using a double sided sticky pad just below the slot. This provides cable retention.

The board can then be lowered into the box (components faced down) on the PCB guides provided. The lid can now be screwed on and the synthesiser is ready for use.

PARTS LIST

Resistors

R1	47k
R2	100k
R3,R4	10k
R5	2k7
R6	100k
R7,R8	33k
R9	10R

All ¼W carbon 5% tolerance

RV1	220k miniature vertical preset
RV2	47k miniature vertical preset
RV3	10k miniature vertical preset

Capacitors

C1	2.2u Tantalum bead
C2	27p Ceramic plate 2%
C3,C7,C9,C11,C12,C13	100n Ceramic 0.2" lead spacing
C4,C5	22n Ceramic
C6,C8	10u Tantalum bead
C10	100u Electrolytic radial lead
C14	1000u Electrolytic radial lead (not larger than 16V W.K.G.)

All 10V working or greater

Semiconductors

IC1	74LS273
IC2	74LS629 or 74LS124 (74LS629 preferable)
IC3	SPO256-AL2
IC4	LM386
TR1	BC108

Miscellaneous

20-way IDC socket – polarised (Dragon 32 only); 26-way IDC socket – polarised (BBC Models A and B only); IC sockets; 8ohm loudspeaker 64mm diameter; Box. Complete kit of parts including pre-drilled box available from L.B. ELECTRONICS.

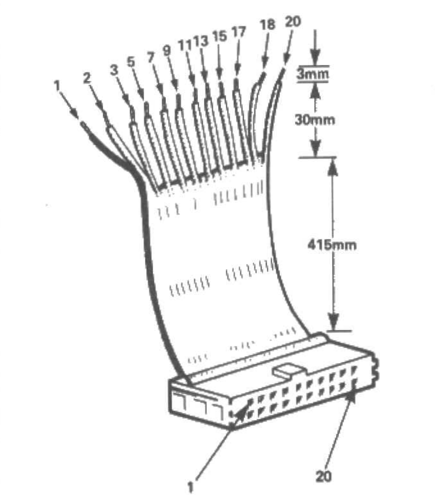


Figure 3. Preparation of the Dragon 32 cable.

Software

To make the synthesiser speak, a character representing the allophone to be spoken must be sent to the printer interface. The commands to do this vary depending upon the host computer. Listed below are the basic commands for the Dragon 32 and the BBC Models A and B:-

Dragon 32

```
  10  PRINT #—2,"character string";
```

The semicolon at the end of the line is required to stop the computer sending a carriage return character.

BBC Models A and B

```
10 VDU2

20 *FX3,26

30 PRINT"character string";
40 *FX3,0
50 VDU3
```

It is difficult to always use a character string in a program because some of the characters are not available on the keyboard. For example, if allophone 0 (pause) had to be sent to the printer (synthesiser) it would be impossible to enter it from the keyboard as this character is not used by the computer.

For this reason it is easier to send numbers directly to the synthesiser (the number being the allophone number). This can be achieved as follows:-

Dragon 32

```
10 PRINT #—2,CHR$(allophone number);
or
10 PRINT #—2,CHR$(I);
```

I being a number variable having the value of the number to be sent to the synthesiser

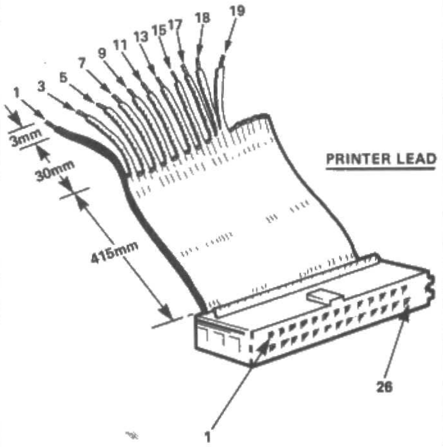


Figure 4. Preparation of the BBC micro's cable.

BBC Models A and B

```
10 VDU2:*FX3,26
20 PRINT CHR$(allophone number);
30 *FX3,0:VDU3
or
10 VDU2:*FX3,26
20 PRINT CHR$(I);
30 *FX3,0:VDU3
```

I is a numeric variable holding the allophone number.

A very simple way to send allophone numbers to the synthesiser is to store the numbers in a DATA statement and then READ and PRINT each number to the synthesiser:-

Dragon 32

```
10 DATA42,23,16,9,49,22,13,51,0.....
     256(allophone numbers,end with 256)

20 READI
30 IF I>255 THEN RESTORE:GOTO20
40 PRINT #—2,CHR$(I);
50 GOTO20
```

BBC Models A and B

```
 5 VDU2:FX3,26
10 DATA8,30,42,19,0....256
20 READI
30 IF I>255 THEN60
40 PRINT CHR$(I);
50 GOTO20
60 *FX3,0:VDU3:END
```

You may have noticed by now that if your press the BREAK key (on the Dragon) before speech has ended, the last sound continues. This must be prevented by making sure that the last allophone is a pause. Pressing the RESET button will also silence the synthesiser.

BBC and Dragon test routines

Program 1 (Dragon 32)

```
  1 REM***DRAGON SPEECH TEST***
  2 REM*HH1,EH,LL,AW,PAUSE,PAUSE***
 10 DATA 27,7,45,32,4,4
 20 PITCH(0)=0:PITCH(1)=64:PITCH(2)=128:PITCH(3)=192
 30 FOR PITCH=0 TO 3
 40 FOR ALLOPHONE=1 TO 6
 50 READ NUMBER
 60 I$=INKEY$
 70 IF I$<>""THEN 20
 80 PRINT#—2,CHR$(NUMBER+PITCH(PITCH));
 90 NEXT ALLOPHONE
100 RESTORE:REM***TO FIRST ALLOPHONE***
110 NEXT PITCH
120 GOTO30
200 PRINT#—2,CHR$(0);
210 PRINT"GOODBYE"
220 END
```

N.B. To STOP program press any key.

Program 2 (BBC Model A or Model B)

```
  1 REM***BBC SPEECH TEST***
  2 REM***HH1,EH,LL,AW,PAUSE,PAUSE***
 10 DIM PITCH(3)
 20 DATA 27,7,45,32,4,4
 30 PITCH(0)=0:PITCH(1)=64:PITCH(2)=128:PITCH(3)=192
 40 VDU2:*FX3,26:REM***DIRECTS OUTPUT
    TO PRINTER (SYNTHESISER)***
 50 FOR PITCH=0 TO 3
 60 FOR ALLOPHONE=1 TO 6
 70 I$=INKEY$(0)
 80 IF I$<>""THEN 200
 90 READ NUMBER
100 PRINTCHR$(NUMBER+PITCH(PITCH));
110 NEXT ALLOPHONE
120 RESTORE:REM***TO FIRST ALLOPHONE***
130 NEXT PITCH
140 GOTO50
200 PRINTCHR$(0);
210 *FX3,0:VDU3:REM***TURNS PRINTER OFF***
220 PRINT"GOODBYE!!!"
230 END
```

N.B. To STOP program press any key.

On the BBC pressing ESCAPE has a more dramatic effect if the commands *FX3,0 and VDU3 have not been reached. Not only does the sound continue but there are also other numbers (the escape message and cursor characters) are sent to the synthesiser and typing on the keyboard only sends characters to the synthesiser and not to the screen.

Normal operation is regained by using:- ctrl C, ctrl L and then type *FX3,0:VDU3 and press the RETURN key.

Another method of sending numbers to the synthesiser is most useful when a spoken word or phrase is to be used many times:-

Dragon 32

```
 5 REM***PROGRAM TO SAY LEFT***
10 DATA45,7,40,13,0
20 FOR ALLOPHONE=1 TO 5
30 READ NUMBER
40 WORD$=WORD$+CHR$(NUMBER)
50 NEXT ALLOPHONE
60 PRINT #—2,WORD$;
```

The word "LEFT" can be repeated by using PRINT#—2,WORD$;

BBC Models A and B

```
  5 REM***PROGRAM TO SAY RIGHT***
 10 DATA39,6,13,0
 20 FOR ALLOPHONE=1 TO 4
 30 READ NUMBER
 40 WORD$=WORD$+CHR$(NUMBER)
 50 NEXT ALLOPHONE
100 DEF PROCright
110 VDU2 *FX3,26
120 PRINT WORD$;
130 VDU3:*FX3,0
140 ENDPROC
```

The word is repeated using:- PROCright.

List of words and corresponding allophones:-

Example Word	Allophone	Allophone Numbers
Alarm	AE,LL,AR,MM	26,45,59,16
Cat	KK1,AE,TT2	42,26,13
Clown	KK1,LL,AW,NN1	42,45,32,11
Collide	KK3,AX,LL,AY,DD1	8,15,45,6,21
Computer	KK1,AO,MM,PP,YY1, UW1,TT2,ER1	42,23,16,9,49,22,13,51
Cookie	KK3,UH,KK1,IY	8,30,42,19
Daughter	DD2,AO,TT2,ER1	33,23,13,51
Down	DD1,AW,NG	21,32
Extent	EH,KK1,SS,TT2,EH, NN,TT2	7,42,55,13,7,7,11,13
Fir	FF,ER2	40,52
For	FF,OR	40,58
Four	FF,OR	40,58
Got	GG2,AO,TT2	61,23,13
Joy	JH,OY	10,5
Left	LL,EH,FF,TT2	45,7,40,13
Letter	LL,EH,TT2,ER1	45,7,13,51
Little	LL,IH,TT2,EL	45,12,13,62
Rat	RR2,AE,TT2	39,26,13
Right	RR2,AY,TT2	39,6,13
Score	SS,KK2,OR	55,41,58
Sister	SS,SS,IH,SS,TT2 ER1	55,55,12,55,13,51
Ski	SS,KK1,IY	55,42,19
Sky	SS,KK1,AY	55,42,6
Square	SS,KK3,WH,XR	55,8,48,47
Story	SS,TT2,OR,EY	55,13,58,20
Thin	TH,EY,NN	29,20,11
Three	TH,ER1,EY	29,51,20
Two	TT2,UW2	17,31
Uncle	AX,NG,KK3,EL	15,44,8,62
Up	AA,PP	24,9
Zero	ZZ,YR,OW	43,60,53

A complete kit of parts for this project is available from LB Electronics at 11 Herdies Road, Hillingdon, Uxbridge, Middlesex (0985 55399).

Also available are the Dragon 32 programs listed above together with software that allows the construction of words and phrases using allophones rather than numbers.

PRINTSPOOLER

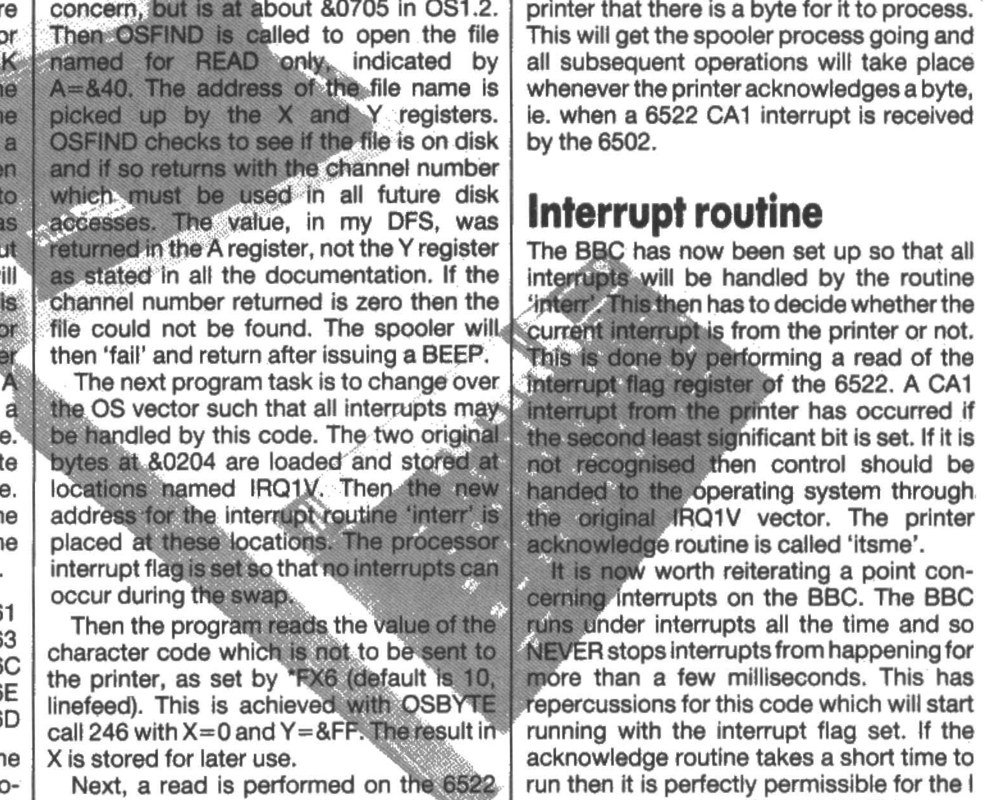
In Part 2 or Chris Cant's BBC printspooler software is a new tube compatible program and a special listing for use with Acornsoft's 'View' wordprocessor.

The fundamental idea of the spooler is that the operation of the parallel interface is controlled completely by the program and all operating system functions are bypassed. A good knowledge of the Centronics printer standard, BBC interrupt control and disk access is required. All operations apart from printer control are done through the operating system so that the code should run under word processors and languages other than BASIC.

A Centronics connector will have a minimum of ten signal lines with additional ground and perhaps power lines. There are eight output data lines and two lines for data handshaking, STROBE bar and ACK bar. When the data byte to be sent to the printer is set up on the data lines, the STROBE bar line is put to a low level for a time of between 1 and 100us and then returned high. This signals to the printer to start processing the data. When it has finished storing it and/or printing it will put the ACK bar line low for 5 to 10us. This will signal to the computer that the printer is ready to accept the next character for printing. In the BBC the A side of the user 6522 VIA controls the printer. The eight A outputs, PA0-PA7, are buffered by a 74LS244 bus driver onto the printer cable. The CA2 output is buffered by a TTL gate and a transistor onto the STROBE bar line. The ACK bar input line goes straight to the CA1 input. Thus, it is possible to control the printer interface using the 6522 registers.

ORA	Output register A	&FE61
DDRA	Data Direction register A	&FE63
PCR	Peripheral Control Register	&FE6C
IER	Interrupt Enable register	&FE6E
IFR	Interrupt Flag register	&FE6D

The program has two main parts. The first part is run to start the spooling process, it opens up the disk file given and then sets up the 6522 registers. Then all subsequent spooler operations are carried out when the BBC is interrupted by the printer when it has finished processing a character. The program intercepts ALL of the BBC interrupts and so has to determine if the interrupt belongs to the printer. If not the operating system is allowed to take control. Otherwise it gets the next character from disk and sends it to the printer. If the disk file is exhausted, then it will be closed, the printer turned off and all OS vectors returned to their previous values.



The first action of the program is to obtain the name of the file which is to be spooled. This will be on the end of user command line which called the program from disk. It is possible to get the address of this using an OSARGS call with A=1 and Y=1. The four-byte address of the position where the rest of the command line is situated is placed at four bytes in zero page indicated by the contents of the X register. Here, X is set to &70 and so the low byte of the address is placed at &70 and the high byte at &71. The actual value is of no real concern, but is at about &0705 in OS1.2. Then OSFIND is called to open the file named for READ only, indicated by A=&40. The address of the file name is picked up by the X and Y registers. OSFIND checks to see if the file is on disk and if so returns with the channel number which must be used in all future disk accesses. The value, in my DFS, was returned in the A register, not the Y register as stated in all the documentation. If the channel number returned is zero then the file could not be found. The spooler will then 'fail' and return after issuing a BEEP.

The next program task is to change over the OS vector such that all interrupts may be handled by this code. The two original bytes at &0204 are loaded and stored at locations named IRQ1V. Then the new address for the interrupt routine 'interr' is placed at these locations. The processor interrupt flag is set so that no interrupts can occur during the swap.

Then the program reads the value of the character code which is not to be sent to the printer, as set by *FX6 (default is 10, linefeed). This is achieved with OSBYTE call 246 with X=0 and Y=&FF. The result in X is stored for later use.

Next, a read is performed on the 6522 ORA. This will clear any old interrupts which may be lodged in the interrupt flag register.

Then, the 6522 PCR register is set up to the desired value. The lower 4 bits have to be set to &E. This sets the CA2 output high and tells the 6522 to interrupt the processor on a CA1 positive transition, and is when the level on CA1 goes from 0 to 1. The upper nibble of the PCR is kept the same as it was before. The old setting of the PCR is stored in 'oldPCR'. The value for the PCR corresponding to high and low levels of CA2 are stored in 'PCRhi' and 'PCRlow' respectively. The PCR pulse output setting is not used for reasons detailed later, despite it seeming to be convenient.

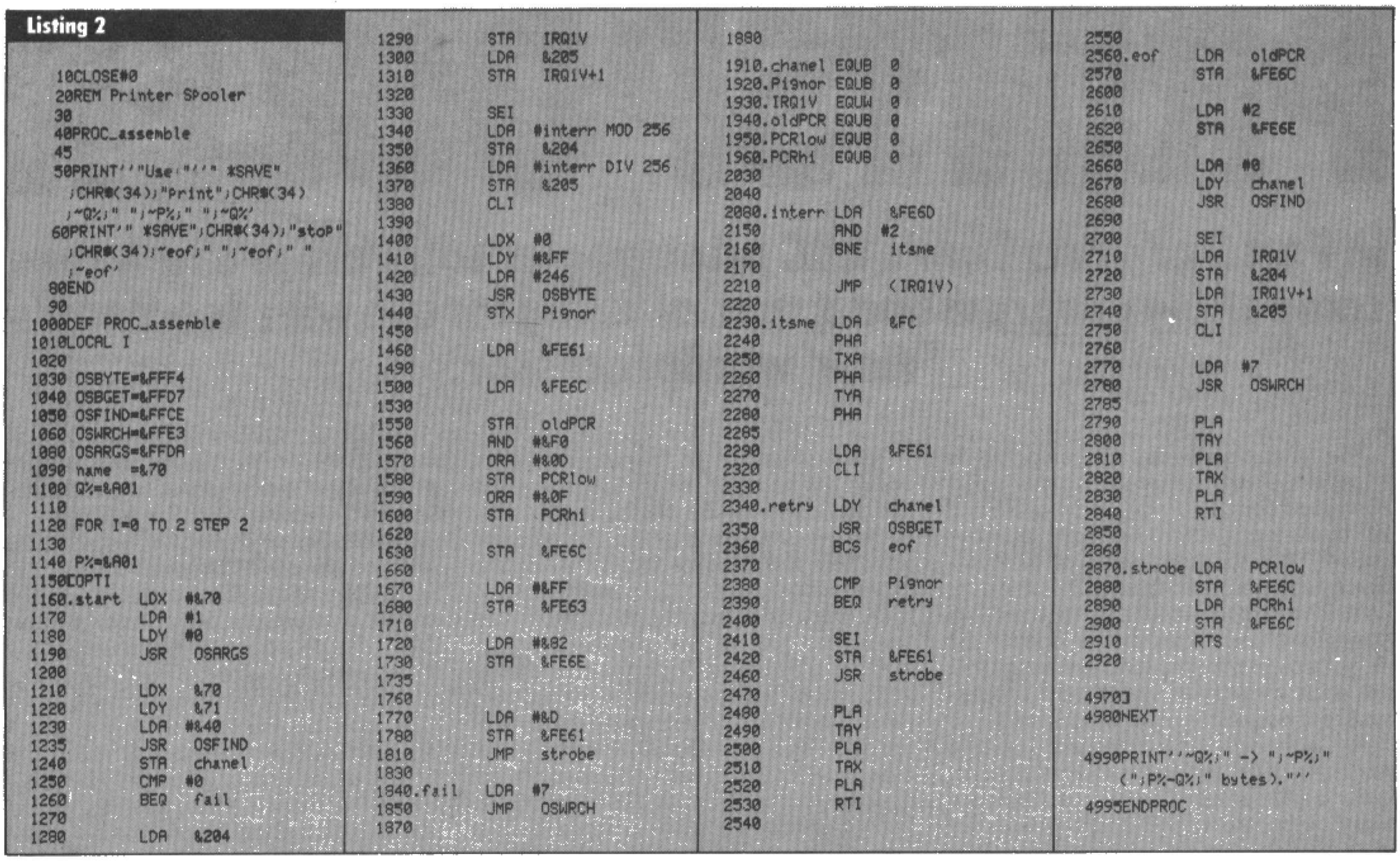
The 6522 data direction register is then set up to make PORTA act as outputs. The interrupt enable register is loaded with &82 to enable interrupts which emanate from CA1 positive edges. Other interrupt enables are not affected. Finally, a carriage return is sent to the printer. A byte of value 13 is stored in the 6522 ORA and routine 'strobe' called. This strobes STROBE low for a microsecond or so to indicate to the printer that there is a byte for it to process. This will get the spooler process going and all subsequent operations will take place whenever the printer acknowledges a byte, ie. when a 6522 CA1 interrupt is received by the 6502.

Interrupt routine

The BBC has now been set up so that all interrupts will be handled by the routine 'interr'. This then has to decide whether the current interrupt is from the printer or not. This is done by performing a read of the interrupt flag register of the 6522. A CA1 interrupt from the printer has occurred if the second least significant bit is set. If it is not recognised then control should be handed to the operating system through the original IRQ1V vector. The printer acknowledge routine is called 'itsme'.

It is now worth reiterating a point concerning interrupts on the BBC. The BBC runs under interrupts all the time and so NEVER stops interrupts from happening for more than a few milliseconds. This has repercussions for this code which will start running with the interrupt flag set. If the acknowledge routine takes a short time to run then it is perfectly permissible for the I flag to remain set until a 6502 RTI instruction is processed. However, for the spooler, a call must be made to the current filing system to get a byte from disk. This may take up to a second if the disk motor has to be started.

So, the first task of the 'itsme' routine is to clear the cause of the interrupt and clear the I flag. This is achieved by performing a read of the 6522 ORA register and doing a CLI. This is also the reason that the handshaking pulse mode output cannot be used here. Both a read and a write with the 6522 PCR set to pulse mode would strobe



Listing 2

```
   10CLOSE#0
   20REM Printer Spooler
   30
   40PROC_assemble
   45
   50PRINT''"Use:"''" *SAVE"
     ;CHR$(34);"print";CHR$(34)
     ;~Q%;" ";~P%;" ";~Q%'
   60PRINT'" *SAVE";CHR$(34);"stop"
     ;CHR$(34);~eof;" ";~eof;" "
     ;~eof'
   80END
   90
 1000DEF PROC_assemble
 1010LOCAL I
 1020
 1030 OSBYTE=&FFF4
 1040 OSBGET=&FFD7
 1050 OSFIND=&FFCE
 1060 OSWRCH=&FFE3
 1080 OSARGS=&FFDA
 1090 name  =&70
 1100 Q%=&A01
 1110
 1120 FOR I=0 TO 2 STEP 2
 1130
 1140 P%=&A01
 1150[OPTI
 1160.start  LDX  #&70
 1170        LDA  #1
 1180        LDY  #0
 1190        JSR   OSARGS
 1200
 1210        LDX   &70
 1220        LDY   &71
 1230        LDA  #&40
 1235        JSR   OSFIND
 1240        STA   chanel
 1250        CMP  #0
 1260        BEQ   fail
 1270
 1280        LDA   &204
 1290        STA   IRQ1V
 1300        LDA   &205
 1310        STA   IRQ1V+1
 1320
 1330        SEI
 1340        LDA  #interr MOD 256
 1350        STA   &204
 1360        LDA  #interr DIV 256
 1370        STA   &205
 1380        CLI
 1390
 1400        LDX  #0
 1410        LDY  #&FF
 1420        LDA  #246
 1430        JSR   OSBYTE
 1440        STX   Pignor
 1450
 1460        LDA   &FE61
 1490
 1500        LDA   &FE6C
 1530
 1550        STA   oldPCR
 1560        AND  #&F0
 1570        ORA  #&0D
 1580        STA   PCRlow
 1590        ORA  #&0F
 1600        STA   PCRhi
 1620
 1630        STA   &FE6C
 1660
 1670        LDA  #&FF
 1680        STA   &FE63
 1710
 1720        LDA  #&82
 1730        STA   &FE6E
 1735
 1760
 1770        LDA  #&D
 1780        STA   &FE61
 1810        JMP   strobe
 1830
 1840.fail   LDA  #7
 1850        JMP   OSWRCH
 1870
 1880
 1910.chanel EQUB  0
 1920.Pignor EQUB  0
 1930.IRQ1V  EQUW  0
 1940.oldPCR EQUB  0
 1950.PCRlow EQUB  0
 1960.PCRhi  EQUB  0
 2030
 2040
 2080.interr LDA   &FE6D
 2150        AND  #2
 2160        BNE   itsme
 2170
 2210        JMP   (IRQ1V)
 2220
 2230.itsme  LDA   &FC
 2240        PHA
 2250        TXA
 2260        PHA
 2270        TYA
 2280        PHA
 2285
 2290        LDA   &FE61
 2320        CLI
 2330
 2340.retry  LDY   chanel
 2350        JSR   OSBGET
 2360        BCS   eof
 2370
 2380        CMP   Pignor
 2390        BEQ   retry
 2400
 2410        SEI
 2420        STA   &FE61
 2460        JSR   strobe
 2470
 2480        PLA
 2490        TAY
 2500        PLA
 2510        TAX
 2520        PLA
 2530        RTI
 2540
 2550
 2560.eof     LDA   oldPCR
 2570         STA   &FE6C
 2600
 2610         LDA  #2
 2620         STA   &FE6E
 2650
 2660         LDA  #0
 2670         LDY   chanel
 2680         JSR   OSFIND
 2690
 2700         SEI
 2710         LDA   IRQ1V
 2720         STA   &204
 2730         LDA   IRQ1V+1
 2740         STA   &205
 2750         CLI
 2760
 2770         LDA  #7
 2780         JSR   OSWRCH
 2785
 2790         PLA
 2800         TAY
 2810         PLA
 2820         TAX
 2830         PLA
 2840         RTI
 2850
 2860
 2870.strobe LDA   PCRlow
 2880         STA   &FE6C
 2890         LDA   PCRhi
 2900         STA   &FE6C
 2910         RTS
 2920

 4970]
 4980NEXT

 4990PRINT''~Q%;" -> ";~P%;"
     (";P%-Q%;" bytes)."''
 4995ENDPROC
```

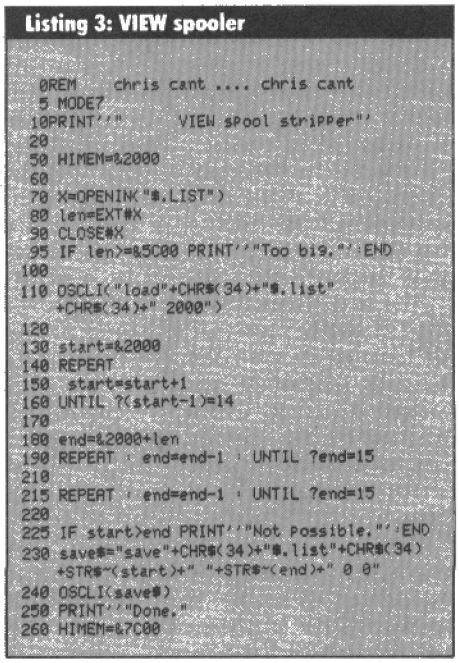
CA2 low. So, this read required to clear the interrupt would also give an erroneous signal to the printer that a byte was ready. Thus, pulse mode is not used and the CA2 line is explicitly strobed low in routine 'strobe'. Also, all the 6502 registers have to be properly saved on the stack in case of further interrupts.

The final section of code occurs when the input file is exhausted and restores the BBC to the state it was in before the spooler was used. The 6522 PCR is reset to its previous value and CA1 interrupts are disabled. The disk file is closed using an OSFIND call with A=0 and Y=channel. The IRQ1V vector is restored to its original value. A BEEP is output to indicate that the routine has finished. All registers are restored and an RTI instruction performed to exit for the last time.

Program enhancements

The program described above will work as a printer spooler but has certain deficiences which are somewhat overcome in **Listing 2.** The first and most obvious change is that the program is now Tube compatible and brings all the benefits of compatibility and longer code. This has other minor effects such as making it necessary to save the X and Y registers for all interrupts, which was not needed before; this is because an OSBYTE call is required to get the 6522 Interrupt Flag register.

These changes also will mean that each interrupt will begin to take longer to process but hopefully should not significantly affect the overall processing speed. It is also worth noting that in these routines we are relying on all operating system routines being written in re-entrant code. This means that it is possible to call a routine from an interrupting program even though it may currently be running in the foreground. This is generally achieved by having any temporary data stored on the stack rather than at specifically addressed locations. Of course there are many more precautions necessary but if in doubt the offending code can be run with the I flag set.



Listing 3: VIEW spooler

```
  0REM    chris cant .... chris cant
  5 MODE7
 10PRINT''"        VIEW spool stripper"'
 20
 50 HIMEM=&2000
 60
 70 X=OPENIN("$.LIST")
 80 len=EXT#X
 90 CLOSE#X
 95 IF len>=&5C00 PRINT''"Too big."':END
100
110 OSCLI("load"+CHR$(34)+"$.list"
    +CHR$(34)+" 2000")
120
130 start=&2000
140 REPEAT
150  start=start+1
160 UNTIL ?(start-1)=14
170
180 end=&2000+len
190 REPEAT : end=end-1 : UNTIL ?end=15
210
215 REPEAT : end=end-1 : UNTIL ?end=15
220
225 IF start>end PRINT''"Not possible."':END
230 save$="save"+CHR$(34)+"$.list"+CHR$(34)
    +STR$~(start)+" "+STR$~(end)+" 0 0"
240 OSCLI(save$)
250 PRINT''"Done."
260 HIMEM=&7C00
```

There is one other small refinement introduced to code. This allows the spooler to be used from WORDWISE which has the nice habit of performing an FS call to close all files after any OS command line is executed, eg. after the spooler is called. This will, of course, cause the program to crash. So, OSFIND is checked to see if a CLOSE#0 call is executed and if so is ignored for the duration of the spool. A specific CLOSE call such as CLOSE#17 will work and will stop the spooler if the channel number is correct.

All OS FS vectors are returned to the values which were present before the spooler was called. The actual code for each redirected vector is fairly complicated, but the alternative was either long and repetitious or self altering code, which really ought not to be encouraged especially when there is no real space restriction. Finally, there is the usual problem of where extra code should be situated in the BBC memory map. Listing 1 assembles to the area which is nominally only used for cassette and RS423 workspace. Listing 2 assembles into the top of the Acorn DFS reserved area. Both may come a cropper if certain commands are executed.

Postscript

The small BASIC program of **listing 3** was written for use with the spooler when used with the VIEW word processor. It will remove the extra garbage added to the text in the *SPOOL file. It deletes up to and including the first occurrance of control code 14 (page mode command). It will delete back from the end until the second occurrance of control code 15 (page mode off). This should leave a virgin VIEW listing. Highlight codes will not be implemented. The program assumes that the text is in a file called $.list. ■

MULTIPRO

James Dick concludes his series on improving computer efficiency by taking a look at multiprocessing techniques and applications.

The biological system taken as an example in the first of these articles is successful because the individuals have a degree of intelligence and can exercise control over their immediate actions. For example, in a column of ants, individuals take care to avoid space about to be occupied by others. Indeed, solving access conflicts is a major task for any designer of multiple processor systems.

The common multiprocessor architectures are shown in Figures 1 to 4. The simplest is shown in **Figure 1.** Here, two processors each have their own core memory but have communal access to peripherals such as disk drives and printers. Users enter tasks into a queue which is serviced by both machines — when one machine's current task terminates, it fetches the next task from the queue. If specialised peripherals (eg. a flatbed plotter) are available only to one machine, then the task's job control environment will specify on which machine the task may execute. This simple linking of processors is referred to as a loosely-coupled system. Assuming machine-specific tasks are rare, the system will process twice as many tasks as a single processor.

"Many microprocessors now come with control lines to facilitate multiprocessing".

A tightly-coupled system is shown in **Figure 2.** As with the loosely-coupled system, both processors share peripherals but now they also share core memory. Access conflicts can be severe depending on the sharing mechanism adopted. A less intimate arrangement is shown in **Figure 3** where only a segment of core is shared. In the event of one processor gaining access to the shared segment, the other may either be locked out or may take alternate clock cycles to obtain access. Many microprocessors now come with control lines to facilitate multiprocessing — the 6809E's BUSY and the 8086's LOCK, for instance.

The final multiprocessing configuration is that of master/slave; shown in **Figure 4,** it is the simplest of those considered and is common where a peripheral has its own processor. For example, drum and flatbed plotters are tediously slow and require complex drive software for the control of stepper motors and pen-lifting solenoids.

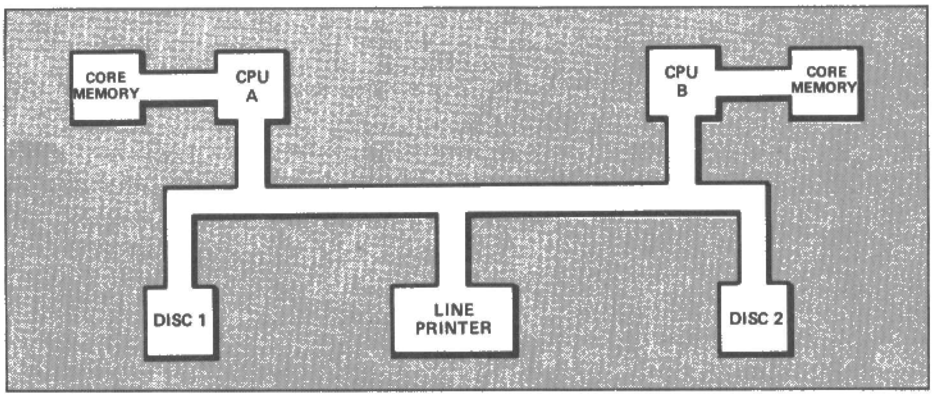


Figure 1. Loosely coupled multiprocessor system.

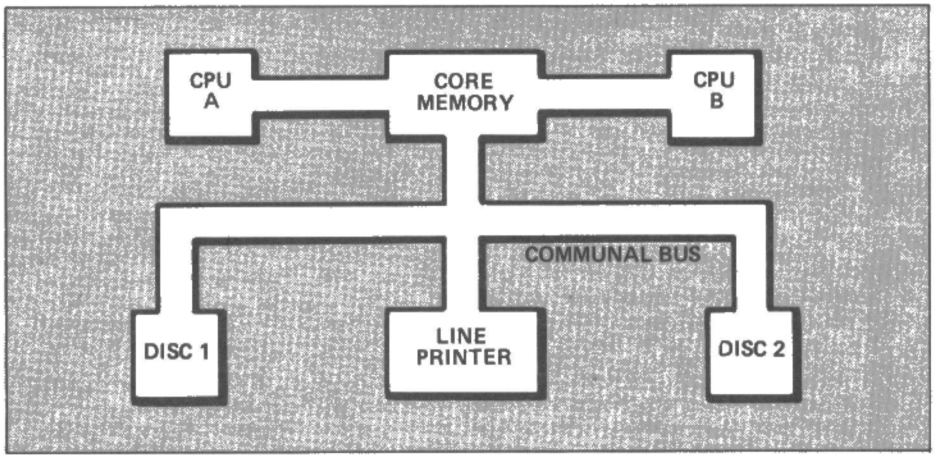


Figure 2. Tightly coupled multiprocessor system.

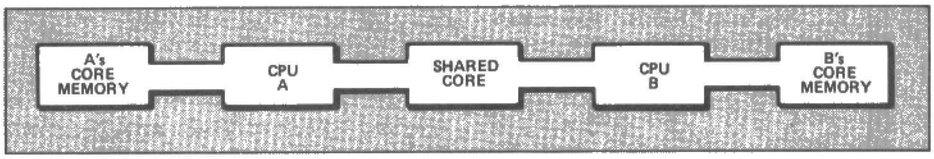


Figure 3. Partly shared memory.

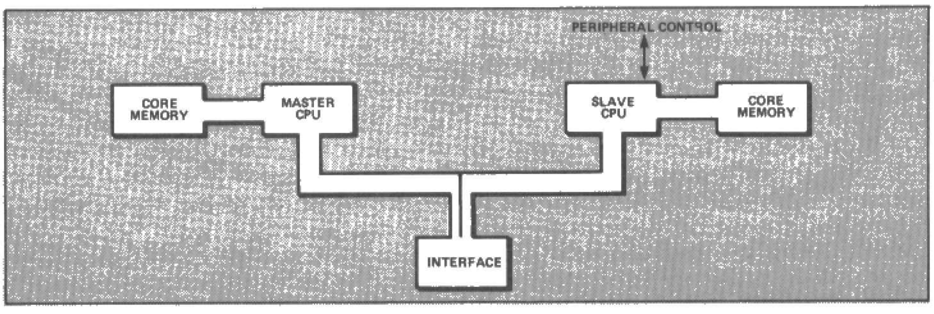


Figure 4. Master-slave system.

OCESSING

A tidy solution is to encode the graphical instructions and dump the coded picture into the slave processor's memory at high speed. The master may then resume execution while the slave produces the plot at its own pace.

Coprocessing

The classical multiprocessor architectures described above all use general purpose processors as elements. Frequently, however, some procedures executed in software by a processor may be more efficiently done if the processor has an internal architecture optimised to meet the needs of the procedures. The most common specialised processors are "arithmetic accelerators": add-on subsystems aimed at improving the speed of mathematical calculations.

Although many microprocessors never perform heavy mathematical work, there are areas where a good FLOPS (floating-point operations per second) rate is important. Image processing for geophysical surveys is one example. **Figure 5** shows how explosions set off from small boreholes at site A create shockwaves which are reflected off density variations in the underlying rocks and received by geophones at site B. Complex analysis of the echoes allows the rock structure to be determined — even limited surveys can take several days of CPU time to analyse.

Many microprocessors perform integer and floating point calculations in software; some modern 8-bit devices (eg. the 6809) have MULtiply instructions. However, multiplication may be carried out in hardware considerably faster than with software algorithms.

The coprocessing approach is best illustrated by an example. Intel have recently announced their 8087 numeric coprocessor to complement the 8086 16-bit microprocessor. The 8087 provides the four common arithmetic functions as well as trigonometric, logarithmic and exponentiation operations on integer operands to 64 bits and real operands to 64 bits — the classical DOUBLE PRECISION data type in Fortran.

In operation, the 8086 and 8087 are connected in parallel. The 8087 is bus compatible with all -86 and -88 devices. Instructions using the 8087 are held in core as ESC instructions; when retrieved by the 8086 they are identified and the 8087 essentially takes control of the system until the numeric operation is done and the processor re-activated. Hence, the 8086 takes up some 25 microseconds later (for an extended precision multiply, for example) with the hard work done.

Coprocessing requires minimal system hardware and is transparent to the programmer. The advantage for the system designer is that a general purpose processor may be built and, when an area of heavy usage is identified, customised for additional performance with minimal fuss. Intel also plan a coprocessor for text processing, input/output, and local area networking.

Software

Excluding the complicated and system-specific supervisor software, most users will be unaware of the multiprogramming nature of their system. Certainly, coprocessors are designed to be transparent to the user. In the past, only designers of dedicated systems became involved in multiprocessing. However, new languages such as ADA are providing complex interaction between program units, allowing a certain degree of simultaneous execution. These features are important for the real-time environment predicted for ADA but will also free the programmer from single threaded execution paths and allow thought structures more akin to daily life with its complex timings and interactions to be modelled on computer systems. ■

> **"Most users will be unaware of the multiprogramming nature of their system".**

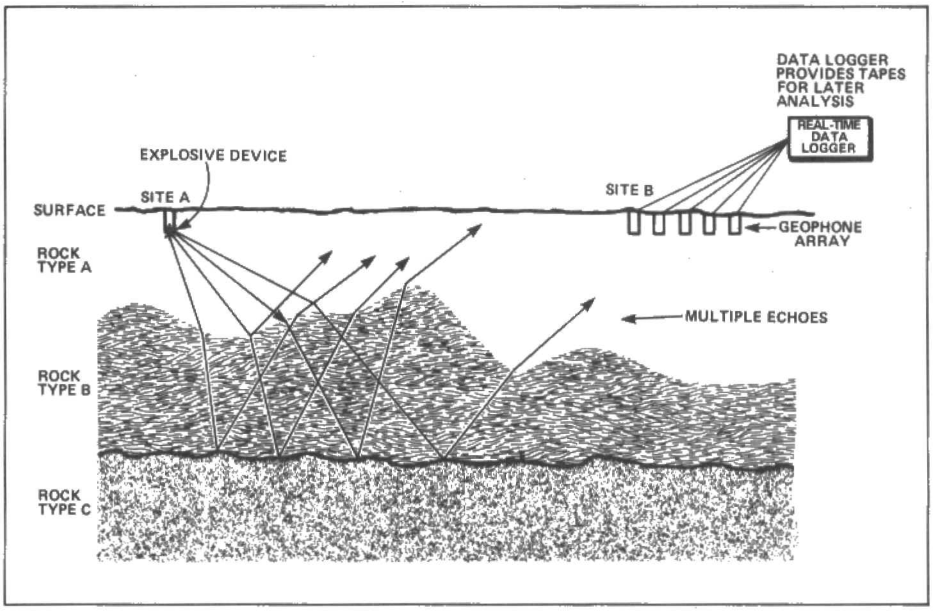


Figure 5. Geophysical surveying.

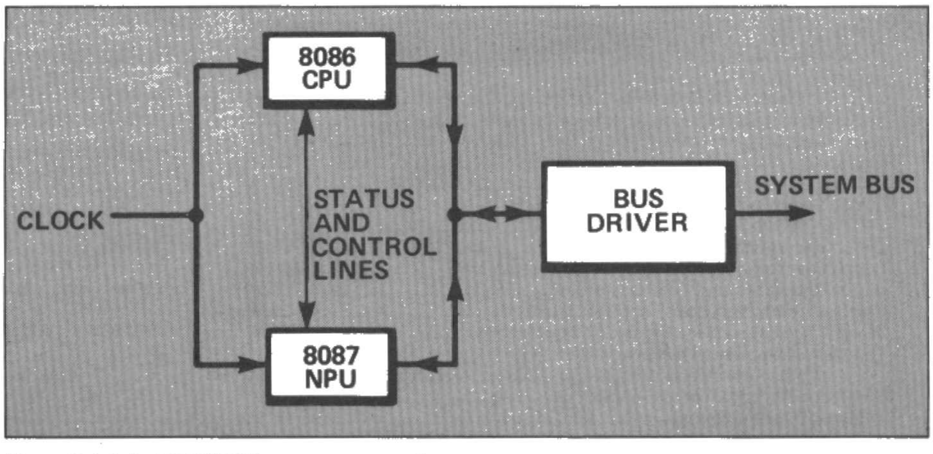


Figure 6. Intel's 8086/8087 coprocessor system.

SAT 16

This month we cover the use and facilities of the SAT-16 MPU developed by Satellite Services for use on the *E&CM* 16-bit computer.

User communication takes place between a standard VDU connected to an RS232 port on the CPU card and the 68000 processor, via the 68701 front-end processor. To understand how communication is effected between the user and the monitor software, a summary of how the 68701 operates must first be understood.

The 68701 has internal software which controls the communication between the user and the 68000 software. To do this, the 68701 has two modes of operation: DIRECT and NON-DIRECT. These function as follows:

"...the powerful 68701 front-end processor has internal software that controls communication between the user and the 68000..."

DIRECT MODE:-

In this mode the 68701 is effectively 'transparent' to the user-68000 link. This means that the 68000 software has direct access to the VDU screen functions and can control cursor positioning, etc. directly. It also means that the 68701 does not 'see' any of the user command facilities available from the VDU but only those from the 68000.

NON-DIRECT MODE:-

In this mode the 68701 directly controls access to the VDU screen. It will arbitrate between the 68000 wishing to output to the screen and the user wishing to enter information which is 'echoed' to the screen. This is done in the following manner:-

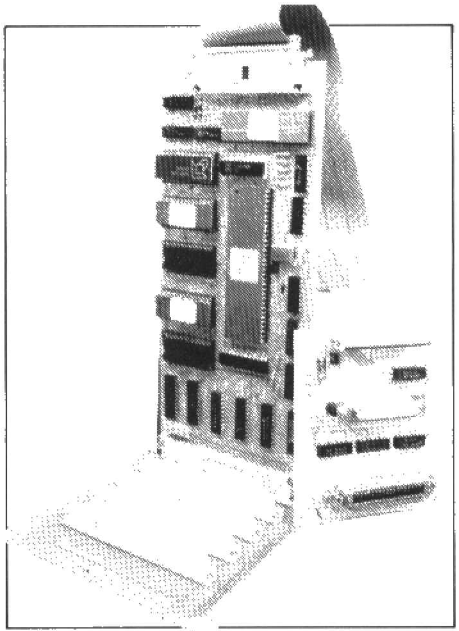
If the 68000 has finished outputting a line to the VDU, (last character sent = LF), the user may request access by hitting any key. When access is granted the 68701 will display a prompt "?>" to the user. The user may then input characters to the system until he hits the 'CR' key. At this point, user access to the system is terminated and if the user requires to input further data he must request access to the system again by hitting any key and waiting for the prompt. This sequence occurs if the 68701 is NOT in continuous mode. Continuous mode, which may be set by user command, will return the prompt to the user when the 68701 is ready for further input. If continuous mode is not set, then, at the end of user input, access is returned to the 68000 and it will be allowed to output to the screen until the user makes another request.

User access requests may be made at any time but access will not be granted until the end of the current line being output from the 68000. Once the user has acquired access to the system the 68701 starts a time-out counter which, if allowed to time-out because the user has not input any further data after the last keystroke, will terminate user access to the system thereby freeing the screen access if the user has forgotten to terminate the input by a 'CR'.

Monitor facilities

SAT16MPU 68000 MONITOR includes the following features:-

- Read and modify any 68000 data register contents.
- Read and modify any 68000 address register contents.
- Modify the User stack pointer.
- Fill a block of memory with a hexadecimal value.
- Move a block of memory to another location.
- Load a program into memory via serial interface.
- Insert/delete/display up to eight breakpoints.
- Start a program loaded into memory.
- Trace a program.
- Read the current time from the RTC.
- Output to a printer port.
- Output current time to the VDU.
- Start a named Operating System.



COMMAND SYNTAX

The format of the Monitor Commands is as follows:-

?CMD (space) (params) (cr)

where-

?	is the prompt
CMD	is the command type
(space)	is the space character
(params)	are qualifiers for the command type used
(cr)	is the return key

All commands must be terminated by the return key for the Monitor to recognise them. The command name may contain up to three characters. The qualifying parameters are in hexadecimal and may be between one and eight digits long. The backspace key may be used whilst typing a command to delete the last character entered.

COMMAND SUMMARY

The following terms apply to the command descriptions:-

adr, adr1, adr2, adr3	represent addresses
dat, dat1, dat2	represent data bytes
n	represents a number
—	represents the minus sign
CTRL	represent the 'control' key
ttttttt	represents text
A	display the contents of all the address registers.
An	display the contents of address register n.
An dat	change the contents of address register n to dat.
B	display all the current breakpoints.
B adr	set a breakpoint at address adr.
B—adr	clear a breakpoint at address adr.
B—	clear all breakpoints.
Note:	This command uses TRAP E vector.
C adr1—adr2,adr3	copy the contents of the memory block between adr1 and adr2 to the same size memory block starting at address adr3.
CMD	List available commands to the VDU.
D	display the contents of all the data registers.
Dn	display the contents of data register n.
Dn dat	change the contents of data register n to dat.
DIR	Output a list of Operating Systems and Utilities found in System Directory.
F adr1—adr2,dat	fill the block of memory between adr1 and adr2 (inclusive) with the data dat.
G	commence a program from the current program counter
G adr	commence a program starting at address adr.
L	load a program from the serial interface.
M adr1—adr2	display the contents of the memory block between adr1 and adr2.
M adr	display the contents of memory at address adr and allow modification as follows:- = the monitor displays the current contents; the user types return: no modification, the monitor displays the contents of the next location. = the monitor displays the current contents; the user enters a new value and then types return to examine the next location. inputting R return, reverses the direction of memory update. Typing T then return terminates the M command.
OPS ttttttt	Start Operating System named ttttttt
PC	display the current contents of the program counter.
PC adr	write address adr to the program counter
PRT ttttttt	Print file named ttttttt. File must have entry in the Directory as a 'data' file.
CTRL Q	interrupt current task and restart monitor. The recognition of this command is dependent upon the status of bit 7 in CNTRLQ in the 68000.
R	display the current contents of all registers.
SR	display the current contents of the status register.
SR dat	change the contents of the status register to dat. Note: 4 bytes must be input but only the MS two are used.
T	execute one instruction from the current program counter and then display the contents of all the MPU registers.
T n	execute n instructions from the current program counter and display the contents of all the MPU registers after each instruction.
TIM	Display the current time on the VDU.
USP	display the current contents of the user stack pointer register.

Breakpoints

The instruction TRAP #$0E will execute the breakpoint function. The breakpoint, once executed, will return control to the Monitor software. The 68000 MPU registers may then be displayed and/or modified before resuming execution of the program from the breakpoint address by using command G. Up to eight breakpoints may be in use at the same time.

The 68000 exception vectors are normally held in EPROM between addresses 000000 and 0003FF. The monitor, however, utilises an indirect jump table located between addresses 004000 and 0043FF to access the vector routines. Since these addresses are located in RAM the user may change the vector pointers to allow use of their own exception routines. The basic table in RAM is initialised by the monitor on start-up. A number of these vectors are already used by the monitor for system routines and the user should take great care if they wish to change these routines. The most dangerous vectors to alter are those relating to external interrupt vectors 1, 2 and 3 since these are used for the hardware handshake between the 68000 and 68701. External interrupt vector 7 is used as a NMI and uses *autovectoring*. External interrupt vector 6 is *non-autovectored* and the 68000 will except a vector number in response to an IACK cycle. External interrupt vector 5 is *autovectored* and therefore if this interrupt can be generated from more than one source then the user must generate software to ascertain which source instigated the interrupt.

A number of system subroutines are available to the user to aid them in implementing their own software. Some of these routines relate to input/output facilities and others are special functions which may be of use in saving code.

INPUT/OUTPUT ROUTINES

These are called by a routine entered by the TRAP instruction. TRAP #$0F followed by a one word function number will execute the following functions:

0 Return to the Monitor.
1 Initialise the input/output device.
2 Output characters from a buffer to the VDU.
3 Input characters into a buffer from the VDU.
4 Load a program.
5 Output characters from a buffer to the PRINTER.

Entry parameters for Input/Output routines, 2, 3 & 5.

Data pointers for the data area to be used for I/O must have been pushed onto the stack in the order indicated in **Figure 1** when the routine is called.

Note: 'buffer pointer' does not have to be the same as 'buffer start' on entry. For output routine it is only necessary to push the 'buffer pointer' and 'buffer end' onto the stack as long as the stack pointer is adjusted by the user's software after exiting from the routine.

MISCELLANEOUS SUBROUTINES

1 Get the current time.
2 Set direct mode in the 68701.
3 Reset direct mode in the 68701.
4 Convert hex to ASCII.
5 Convert ASCII to hex.
6 Transfer a block of memory.

GET CURRENT TIME

This subroutine will get the current time from the RTC in the 68701 and put it into TIMBUFF (004408) in the following format:

TIMBUFF	hours in BCD code
TIMBUFF+1	minutes in BCD code
TIMBUFF+2	seconds in BCD code
TIMBUFF+3	1/100 seconds in binary

On exit ACIAS bit 3 will reflect the success of the time call. The routine is called by "JSR GETTIME". ($0E00).

SET DIRECT MODE

This routine sets "direct" mode in the 68701. The routine is called by "JSR DIRSET". ($0F00).

RESET DIRECT MODE

This routine resets "direct" mode in the 68701. The routine is called by "JSR DIRRES". ($0F24).

CONVERT HEX-ASCII

This routine converts a four byte value 'VALUE' into eight ASCII hexadecimal digits and stores them into a buffer starting at address 'BUFPTR'. The entry parameters for this routine must be stored on the stack before calling in the sequence indicated in **Figure 2.**

This routine is called by "JSR OUTHEX". ($100C).

CONVERT ASCII-HEX

This routine converts eight ASCII hexadecimal characters located in a buffer starting at address BUFPTR into a four byte value 'VALUE' on the stack. Entry parameters for this routine must be initialised on to the stack in the sequence indicated in **Figure 3.**

This routine is entered by "JSR INHEX". ($1010).

BLOCK TRANSFER

This routine transfers a block of memory starting at address BLKSTR and ending at address BLKEND into a buffer area starting at address BUFPTR. Entry parameters for this routine must be initialised onto the stack, before calling, in the sequence indicated in **Figure 4.**

This routine is entered by "JSR MOVEB". ($1014).

Front-end processor facilities

The 68701 front-end processor provides the means of communicating between the 68000 CPU and the user VDU and printer. The link protocol is set up in the following manner. ASCII data sent from the 68000 to the 68701 with the most significant bit reset (bit7=0) will be considered as data for the user VDU. ASCII data sent from the 68000 to the 68701 with the most significant bit set (bit7=1) will be considered as data for the printer. All ASCII data sent from the user VDU via the 68701 to the 68000 will have the most significant bit reset (bit7=0). Data received from the 68701 by the 68000 with bit7=1 will be considered to be a copy of the 68701 Status Register (SAT-2).

A number of data sequences are reserved and used as control codes to the 68701. These make certain utilities available to the user. Control sequences can be accessed from both the 68000 and via the user VDU but certain restrictions apply depending upon the source of the request.

The data sequences that generate control codes are as follows:

*/n this is the basic sequence to generate a control function where:-
* is the asterix character (ascii code 2A)
/ is the slash character (ascii code 2F)
n is the function code as follows:-

FUNCTION CODES (from the 68000 to the 68701)

? send the current time to the 68000. The time is returned as 7 bytes in the following format:-

bytes0,1	*(ascii 2A)
byte2	hours in BCD code
byte3	minutes in BCD code
byte4	seconds in BCD code
byte5	1/100 seconds in binary code
byte6	CR (ascii 0D)

H send a "request for printer" message to the user via the VDU. This will output a message to the user VDU asking for the printer to be put on-line. The 68000 software may examine bits in the 68701 Status Register (SAT-2) to determine the status of the printer.

D sets "direct" mode in the 68701. This is the mode that is set on start-up by the 68000. This mode makes the 68701 transparent to the system and allows the 68000 software to directly control the VDU screen. Note: in this mode most of the control facilities from the VDU to the 68701 are inhibited.

Z resets "direct" mode in the 68701. This enables the user facilities available from the VDU but also controls the screen access between the 68000 and the user. ie If the 68000 is outputting to the VDU the user must first request access by hitting any key and then wait for the prompt "?>" before being able to input to the system. There is a time-out for the user prompt so that the 68000 does not wait for ever if it needs to output information to the screen. The user is granted access after the next carriage return is received from the 68000. If in NON-DIRECT mode then the 68000 should not echo input characters as these will already have been echoed by the 68701.

Any other command code is illegal and the 68701 will return STAT2 to the 68000 with bit4 (illegal command) set.

FUNCTION CODES (from the VDU to the 68701)

C Toggle continuous mode bit. Continuous mode, when set, automatically puts a prompt out to the user when the VDU can accept input.

G start RTC.

S stop RTC.

R reset RTC. The RTC running/stopped condition is not affected by this command.

I inhibit printer. This sets the status bit in STAT2 to indicate that the user has removed the printer from the system.

P restore printer. Cancels the I command.

T send current time to the VDU screen.

K request the user for a new time. User is prompted to input the time.

All other commands are illegal and the 68701 will output "INVAL" to the VDU screen.

68701 TO 68000 HARDWARE/ SOFTWARE PROTOCOL

This appendix describes the way in which data from the 68701 to the 68000 is interpreted and handled. This section should be thoroughly understood by the user if he wishes to handle his own input data since there are a number of factors which determine what the 68000 will do with data that it receives over this link.

The main factors that decide the handling of data from the 68701 are as follows:-

a — The state of bit 7 in CNTRLQ. (address 4405)
b — The state of bits 4 and 5 in ACIAS. (address 4401)
c — The state of bit 7 in the data being received.
d — The actual data itself being received.

The above conditions are checked in the following order and are acted upon as described.

1. Is the character code Hex 11?
   If NO then skip to 3.
2. Is bit 7 set in CNTRLQ?
   If YES then CLEAR THE SYSTEM AND RESTART MONITOR.

```
 .--------------------.
 |    buffer start    |
 *--------------------*<---SP + 8
 |   buffer pointer   |
 *--------------------*<---SP + 4
 |     buffer end     |
 '--------------------'<---STACK POINTER
```

Figure 1.

```
 .--------------------.
 |    buffer start    |
 *--------------------*<---SP + 12
 |   buffer pointer   |
 *--------------------*<---SP + 8
 |     buffer end     |
 *--------------------*<---SP + 4
 |       value        |
 '--------------------'<---STACK POINTER
```

Figure 2.

```
 .--------------------.
 |    buffer start    |
 *--------------------*<---SP + 12
 |   buffer pointer   |
 *--------------------*<---SP + 8
 |     buffer end     |
 *--------------------*<---SP + 4
 |       value        |
 '--------------------'<---STACK POINTER
```

Figure 3.

```
 .--------------------.
 |    buffer start    |
 *--------------------*<---SP + 16
 |   buffer pointer   |
 *--------------------*<---SP + 12
 |     buffer end     |
 *--------------------*<---SP + 8
 |    block start     |
 *--------------------*<---SP + 4
 |     block end      |
 '--------------------'<- -STACK POINTER
```

Figure 4.

3. Is bit 7 set in the character code?
   If YES then store character in STAT2, set bit 4 in ACIAS and exit.
4. Is bit 5 set in ACIAS?
   If YES then put character in ACIAD, set bit 0 in ACIAS and exit.
5. Put character in ACIAD2 and TRAP D
   Note: TRAP D is not implemented in this monitor and therefore the character would normally be lost. This then provides a means where the user can accept extra input from the VDU without calling the input handler routine via TRAP F and sitting idle until data is input.

Notes on Input Handling Routines

The user has control over the preceeding factors when implementing input from the VDU for his own software. The monitor uses the above factors in the following manner:-

When the monitor has issued a prompt and is waiting for the user to reply, ACIAS bit 5 is set. This bit is reset when a 'CR' is detected since this also signifies the end of the users input. On start-up of the monitor, CNTRLQ bit 7 is always set.

Appendix

STATUS REGISTER

68701 STAT2 (address 004407 in 68000)
bit 7 – ALWAYS ONE
6 – RTC INHIBITED
5 – PRINT BUFFER FULL
4 – ILLEGAL COMMAND
3 – PRINTER REQUESTED BY 68000
2 – VDU BUFFER FULL
1 – PRINTER PORT FAILURE
0 – PRINTER DISCONNECTED BY USER

68000 ACIAS (address 004401)
bit 7 – PRINTER DATA NOT SENT SUCCESSFULLY
6 – DIRECT MODE SET
5 – INPUT EXPECTED
4 – STATUS RECEIVED
3 – TIME VALID
2 – RESERVED
1 – OUTPUT BUFFER EMPTY
0 – INPUT BUFFER FULL

68000 CNTRLQ (address 004405)
bit 7— 1=CTRL Q ENABLED
0=CTRL Q DISABLED
BITS 6-0 RESERVED

TIMBUFF format (address 004408 on)
TIMBUFF – hours in BCD
TIMBUFF+1– minutes in BCD
TIMBUFF+2– seconds in BCD
TIMBUFF+3– 1/100 seconds in binary

ACIAD (address 004403)
Expected data from 68701

ACIAD2 (address 004411)
Unexpected data from 68701 ■

READERS' EXCHANGE

A free service for readers wishing to buy, sell or exchange computer equipment. Please send your advertisement, clearly printed, together with your name and address to: Readers' Exchange, Electronics & Computing Monthly, 155 Farringdon Road, London EC1R 3AD.

FOR SALE. TRS-80 Model One, Level Two 16K RAM, complete with green monitor, cassette, Aculab floppy tape system, Centronics printer, light pen, hundreds of programs including Scripsit Word Processor, Databases and games. Also sound amplifier and books. Bargain at £350. B. L. Anstead, 22 Anthony Road, Welling, Kent DA16 3EJ. Tel: 01-854 4348 evenings.

WANTED To buy or photocopy, details of pin-outs of edge connector on a pair of Olivetti 8" floppy disc units, or sell. Offers (complete with firmware disc). Blyth 361094 (North). Mr. Coulson, 9 x 7th Avenue, Blyth, Northumberland NE24 2SZ.

FOR SALE, BBC micro Model B + cassette recorder + special printer cable + set of joysticks + games software +database software + word processing software + BASIC course in book form and cassette + book + 1.2 ROM. Sell for £430. Phone 061 368 7145. David Boden, 41 Silverton Close, Hattersley, Hyde, Cheshire.

US COMMODORE 64 enthusiast desires contact and correspondence with other '64' users. Wish to swap information, books, software, etc. Let's have an international users group! Write to: Paul D. Filiault, 816 County Street, #16, Taunton, MA, USA 01780.

WANTED: Information from anyone who got the ZX Spectrum sounds board (*E&CM* August '83) to work. Will gratefully pay any expenses etc. for replying. Tony Healy, 98 Elm Park, Clonmel, Eire.

FOR SALE: BBC B 1.2os, EPROM Blower, Light Pen, Wordwise (word processing), EXMON (ext machine code monitor). Software on tape: Educ and games worth over £100. Tons of books and magazine. Everything worth over £650, bargain at £450 ono. Might consider a swap, MTX500 or 512 plus extra. H. S. Hothi, 25 Leeds Road, Slough, Berks. Tel: 72535.

BBC Wordwise ROM as new, will swop for an 8271 disc controller. Vincent Tel: (0524) 411780 Morecambe after 6 p.m.

DRAGON 32 micro fro sale, 4 books, 2 Dragon joysticks, over £105 worth of software. Altogether worth £357. Sell for £250 ono. Phone Dorking 888426, evenings.

68705 EPROM blower

A 68705 project by John Williams which allows EPROMs to be programmed on any computer equipped with RS232.

There have been several designs published in *E&CM* for EPROM programmers to interface with specific microcomputers. These are fine for their purpose but have to be discarded if the user wishes to upgrade his computer system to use a new model. The publishing of Richard Whitlock's recent series in *E&CM* on the one chip Motorola 68705P3 prompted the author to design an EPROM programmer to be compatible with any present or future computer provided with an RS232 interface. The requirement was for a unit suitable for programming and listing a range of EPROMs. To extend its useful life it was made to handle the larger EPROMs which are now becoming available. The unit is not able to actually run the program stored because this requires access to the computer's microprocessor bus and is therefore machine dependent. The amount of control software required in the control computer has been kept to an absolute minimum by the choice of a versatile programmer control syntax. With the addition of only one control character, EPROMs can be programmed or verified directly from almost any hexadecimal code dump routine likely to be found in the host computer. Code developed in RAM can be easily prommed by this means.

EPROM ranges

There are two ranges of EPROMs commonly used today. The one chosen for this design is probably the more popular; it is the 27xx series starting with the 2716 and progressing to the 27265. **Figure 1** shows the similarity between family members. The other group includes the 2532 and 2564. These differ in some pin allocations and programming requirements and to simplify the programmer design have been excluded from this project. This unit will also code with the TMS2516 which is equivalent to the 2716 but will not drive the TMS2716 which is a 3 rail EPROM.

Apart from the pin differences, the other variation in the 27xx series, shown in **Figure 2,** is the value of the extra voltage required for programming. All the manufacturers the author has encountered specify 25 volts to program their 2716s and 21 volts for the 2764 and 27128. The situation is not so clear for the 2732. Most low cost versions available from the hobbyist outlets require 25 volts, however Intel's 2732A, which has completely replaced their 2732, requires 21 volts and this voltage may be specified by some other manufacturers. The new Intel 27256 requires 13 volts, though it will be some time before the price drops sufficiently to make it suitable for home use.

Type	Voltage
2716	25
2732	25
2732A	21
2764	21
27128	21
27256	13

Figure 2. Programming Supply.

PIN NAMES

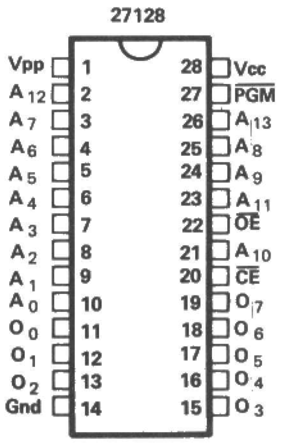
A_0-A_{13}	ADDRESSES
$\overline{CE}$	CHIP ENABLE
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS
$\overline{PGM}$	PROGRAM
NC	NO CONNECT

Programming procedure

In order to program an EPROM it is necessary to set up and hold the required logic states on its address pins and to feed the desired code for that address to the 8 data pins. With the programming voltage present and with other control pins correctly set, a pulse of 50ms is applied to a specified pin. The selected address will then become programmed and any further addresses can be subsequently programmed in the same way. The actual control signals and pins to which they are applied vary between the types of EPROM used. **Figure 3** shows, for each type, the waveforms required for both programming and reading back information. Only one programming pulse is shown but where more than address is to be programmed the 50ms pulse is repeated for each address while the other lines remain steady. It should be noted that the pin numbers shown are for the 28 pin socket and the corresponding pin numbers for a 24 pin IC are shown in brackets where relevant. When a 24 pin IC is used it must be plugged into the 28 pin socket so as to leave pins 1, 2, 27 and 28 vacant.

Data sheets show that all but the 2732 have the facility to verify the programmed data with the high voltage still applied. This can be utilised in a programmer to save time in the overall process of programming and verifying. However it would complicate the overall design of this programmer, especially as it would have to deal differently with the 2732. As this is intended mainly as a hobbyist's programmer it was felt that speed could be sacrificed for simplicity. Provision was therefore made for verifying the EPROM content in a

27256	2764	2732A	2716
Vpp	Vpp		
A_{12}	A_{12}		
A_7	A_7	A_7	A_7
A_6	A_6	A_6	A_6
A_5	A_5	A_5	A_5
A_4	A_4	A_4	A_4
A_3	A_3	A_3	A_3
A_2	A_2	A_2	A_2
A_1	A_1	A_1	A_1
A_0	A_0	A_0	A_0
O_0	O_0	O_0	O_0
O_1	O_1	O_1	O_1
O_2	O_2	O_2	O_2
Gnd	Gnd	Gnd	Gnd



2716	2732A	2764	27256
		Vcc	Vcc
		$\overline{PGM}$	A_{14}
Vcc	Vcc	NC	A_{13}
A_8	A_8	A_8	A_8
A_9	A_9	A_9	A_9
Vpp	A_{11}	A_{11}	A_{11}
$\overline{OE}$	$\overline{OE}$/Vpp	$\overline{OE}$	$\overline{OE}$
A_{10}	A_{10}	A_{10}	A_{10}
$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$
O_7	O_7	O_7	O_7
O_6	O_6	O_6	O_6
O_5	O_5	O_5	O_5
O_4	O_4	O_4	O_4
O_3	O_3	O_3	O_3

Figure 1.

NOTE: Intel "Universal Site" compatible EPROM pin configurations are shown in the blocks adjacent to the 27128 pins

separate pass from programming and to do this with only the 5 volts present. The design also makes no attempt to utilise the optional programming algorithms available for use with the large capacity EPROMs. If used the algorithm allows the programming pulse to be shortened from the standard 50s, provided that the effectiveness of the programming is measured by a specified technique. Again they were considered a luxury for a hobby machine. As a result the time taken to program and verify an entire 2764 is about 30 minutes with a 300 Baud interface and is proportional for other size EPROMs. 300 Baud was chosen for the design as it allows operation without any handshake being required over the RS232 line. The handshake lines are however provided and a simple modification to the code allows the unit to be used with a handshaken 1200 Baud interface. This significantly speeds up the operation, particularly when listing and verifying.

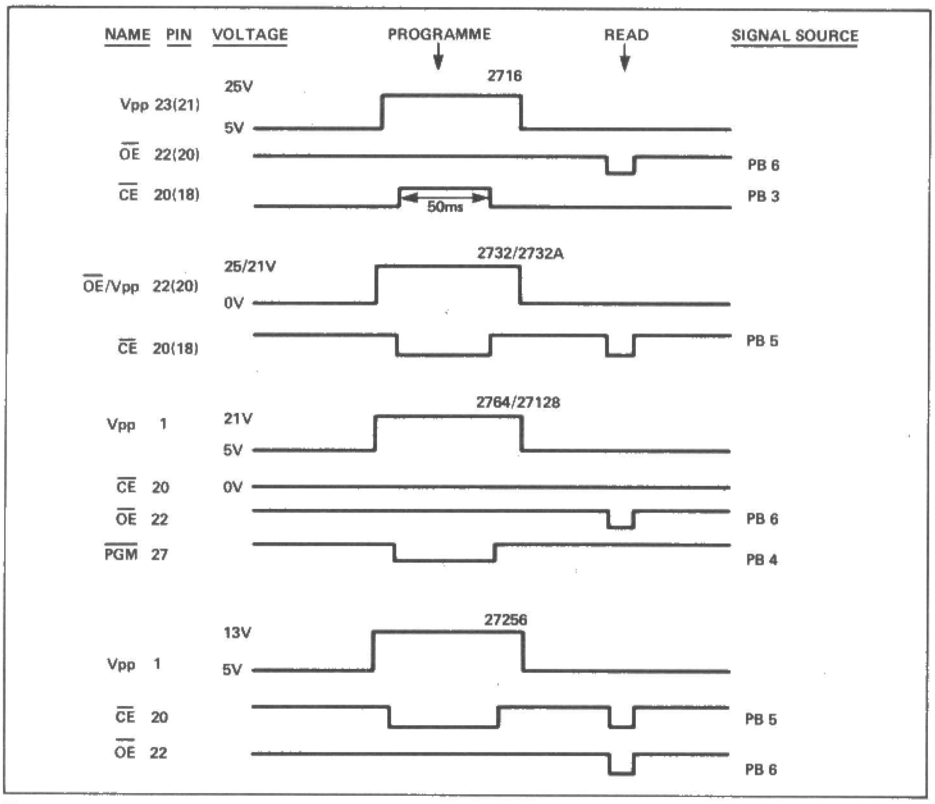


Figure 3. EPROM read/write waveforms.

Pin Numbers	1	2	3	4	5	6
Free Socket	EPROM 20	EPROM 22	EPROM 23	EPROM 26	EPROM 27	EPROM D5
2716 Plug	PB3	PB6	Vpp	Vcc	Nc	Vcc
2732/A Plug	PB5	Vpp	A11	Vcc	Nc	Nc
2764/128 Plug	0V	PB6	A11	A13*	PB4	Vcc
27256 Plug	PB5	PB6	A11	A13	A14	Vcc

* Not connected internally on 2764

Nc No connection

Figure 5. Connection details of EPROM selection plugs and socket.

Circuit description

The 68705P3 integrated circuit which forms the heart of the design contains microprocessor, RAM, EPROM, timing circuits and input/output ports. For details of the chip the reader should refer to Richard Whitlock's series starting with the October 1983 issue of *E&CM*. For operation the microprocessor requires only a 5 volt power supply, the connection of a crystal and capacitor, and also a power-on reset capacitor. The remainder of the circuitry of **Figure 4** is dedicated to this programmer application. The 68705P3 has 20 input/output lines; these have to drive the address, data, and control pins of the EPROM and also to provide the RS232 interface. As this requirement totals more than 20 lines some form of expansion technique is required. It was decided to allocate the 8 port A lines to be input or output to the EPROM data pins. The interrupt input and three of the port C lines are used to provide the RS232A inputs and outputs while seven of the port B lines provide control to the EPROM and drive indicator LEDs. The EPROM address is generated from only one port C and one port B line. The address, of potentially up to 16 bits, is provided from two serial in-parallel out shift registers connected in series. The sixteen bits of the address are output sequentially from PB7 with the most significant bit sent first. While PB7 sets the data, PC3 is pulsed a total of 16 times in order to clock it through the shift register to assemble the address. This method was chosen in preference to two 8-bit latches as the wiring is substantially simplified at the expense of only a small additional amount of software.

The RS232 input and output buffers are provided by IC1 and IC2. The input is a standard MC1489 RS232 line receiver, it will accept any voltage from −12 to 0.75V as a logic low and 1.5 to 12V as logic high. Interfaces normally use an MC1488 as a driver, however, a standard TTL gate is used in this design to reduce the number of power rails needed. Over a distance of a few feet this is satisfactory for driving the MC1489 line receiver found in most RS232 equipment. The circuit design has however been made in such a way that IC2 can be easily replaced by a MC1488 driver giving proper RS232 levels if required. To do this pin 14 of IC2 is isolated from 5V and connected to a +9 to +12 volt supply while pin 1 is connected to an equal negative supply. The encoding between serial and parallel formats is all done by the microprocessor software.

Apart from the programming voltage, a total of four different TTL waveforms are needed to satisfy the programming and reading requirement of the 5 different EPROMs. These waveforms shown in **Figure 3** are generated by software and output from ports PB3 to PB6. The microprocessor produces all four waveforms regardless of the type of EPROM in use and a simple jumper plug makes the required connections to the selected EPROM. The jumper plug also takes account of any other connection variations. Analysis of the pin allocations of Figure 1 and the waveforms of Figure 3 shows that pins 20, 22, 23, 26, and 27 require different functions between types. These IC pins are connected to a 6 way socket on a flying lead. To select the required EPROM this socket is inserted onto one of the four 6-way plugs, the pins of which are wired up to have the correct signals to go to the appropriate IC pins. The detailed requirements for the plug connections are shown in **Figure 5.**

The 25, 21, or 13 volt programming supply Vpp is applied to the EPROM only during programming and is controlled from PB2 via transistors TR1 and TR2. All but the 2732 require the Vpp pin to go to 5 volts for reading. This is achieved within the tolerance specified for the IC by diode D5 via the 6th pin of the jumper socket. The corresponding pins connect the anode of the diode to +5 volts. The diode is unconnected for the 2732 and the resistor R7 pulls the line down to 0V. The capacitor C6 suppresses possible damaging transients on the Vpp line and is of the value recommended by Intel. The LED D4 indicates whenever the programming voltage is switched through to the EPROM.

Next month: Construction and software details.

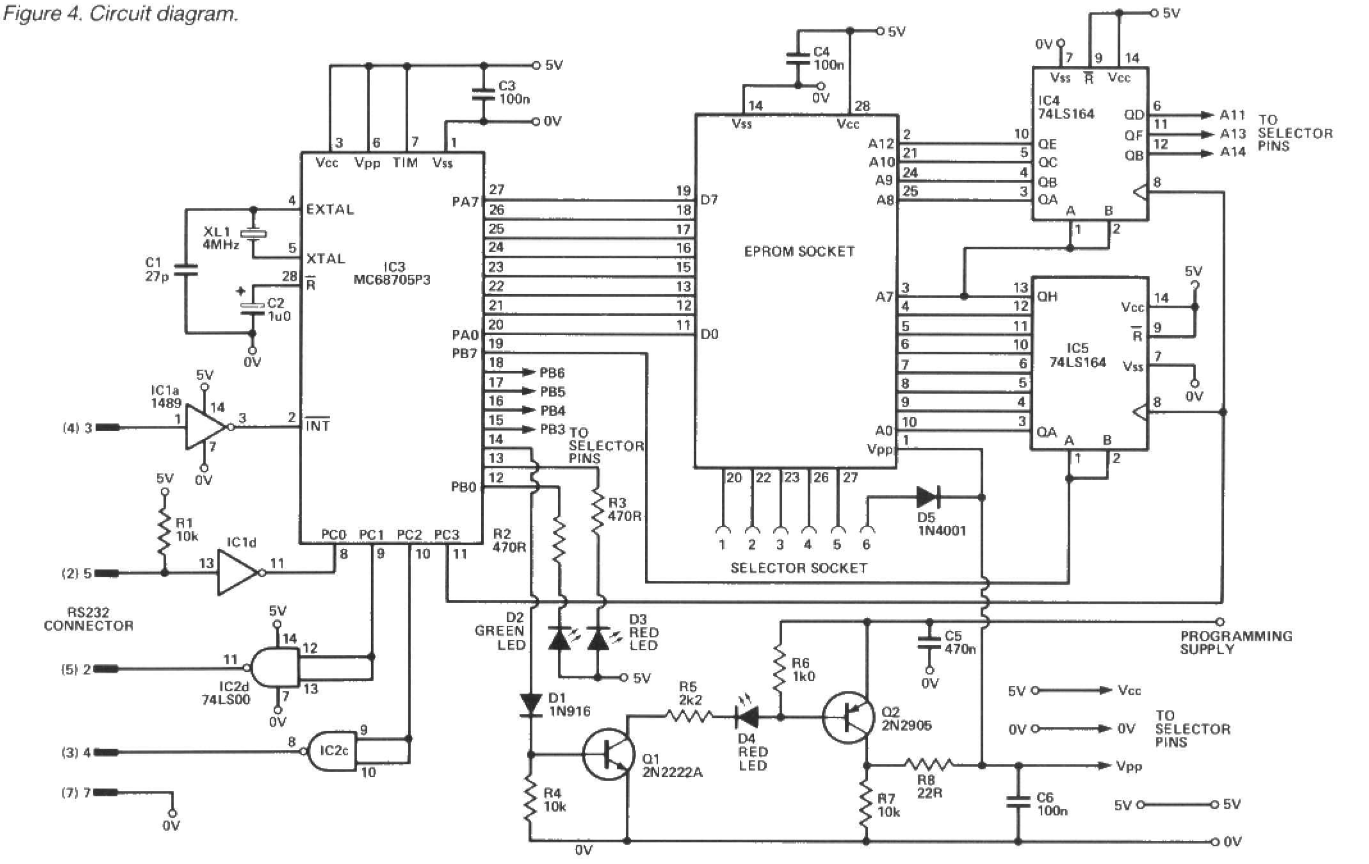


Figure 4. Circuit diagram.

Born again Oric

Is ORIC's new computer merely an ORIC 1 with a facelift? After a thorough benchtest Mike James thinks not.

The Oric Atmos is a difficult machine to review because although it is being presented, and for the most part being treated, as a new machine it has a great many similarities to the original Oric 1. Indeed the best description of the Atmos is as an Oric 1 with a new keyboard and a new ROM. However, simply to dismiss the machine as a mark two Oric is also incorrect as many of the important features of a computer are determined by its software.

The two main groups of people interested in details of the Atmos are its potential buyers and current owners of Oric 1's. The information needed by both groups is best provided by a brief description of hardware and software that makes up the Atmos and more detailed comments on how it differs from the Oric 1. If at the end of this 'micro' review you still feel in need of more information then refer to the full review of the Oric 1 in the April 1983 issue of *E&CM.*

The hardware

The Atmos is a single speed 6502-based home computer offering high and low resolution graphics and three channel sound. It is a 48K machine and costs £170. The main feature of the Atmos that sets it apart from other micros is its provision of teletext compatible graphics. As well as the simple provision of the standard teletext block graphics, something that is easy to provide on any micro with enough user-defined characters; this also brings with it the teletext way of controlling colour – serial attributes. Teletext is essentially a two-colour system with local control over which two colours are displayed at any given character location. The colours are selected by storing special 'attribute' codes in screen memory locations. That is, a character location can either display a text or graphics character or it can hold a code that effects the way other characters are displayed. The attribute codes are all displayed as blanks or spaces and they effect all of the characters to their right and on the same line. The reason for this strange mechanism for colour control is not difficult to appreciate when you consider the way that a teletext display is transmitted and the need for simplicity in teletext receivers, but it does make things a little more difficult than they need be in a home computer. It's not that it is impossible to produce good low-resolution colour graphics using serial attributes – a few minutes spent looking at a teletext transmission will soon prove otherwise – but the restrictions introduced by storing attributes on the screen take some time to get used to. Within the restrictions of serial attributes the text and low-resolution display is more than adequate with 28 lines of 40 characters, two completely user-definable character sets and the ability to use eight different colours, flashing and double height characters. *(See the article 'Graphics Designer' on Page 46 for a further discussion of Teletext Graphics).*

The same serial attribute method of controlling colour is used in the 200 by 240 pixel resolution high resolution display mode. There are also two text lines displayed at the bottom of the screen that are ideal for messages and input prompts but the software also makes it possible to print characters at any position on the screen for graph labelling etc. The use of serial attributes seems a little more out of place in high resolution graphics as it involves the idea of storing attribute codes in screen memory locations. Once again an attribute code is displayed as a blank or space and it affects the colour of all the pixels to the right and in the same row. Using the full eight colours to good effect in high resolution graphics is also difficult but not impossible, for more information see "The High Resolution Oric" by S. M. Gee in the January 1984 issue of *E&CM.*

Atmos sound is loud and easy to use. With three tone and one noise channel it is also versatile. Three note chords and sound effects are no problem. The topic of the Oric's sound capabilities was discussed in some detail in last August's *E&CM.*

As far as hardware compatibility with the Oric 1 goes the ATMOS appears to **be** an Oric 1. Indeed when I opened the review model the printed circuit board was marked 'ORIC 1 issue 4'. Even the case is closely modelled on the original ORIC 1 with the addition of a very smart new red and black top and standard typewriter style keyboard. The new keyboard makes the Atmos much more suitable for applications such as word processing than the Oric 1 but it is still not up to professional standards.

Atmos BASIC

The main new component of the Atmos is its 16K ROM. ATMOS BASIC is best described as a no frills Microsoft BASIC. It includes IF . . . THEN . . . ELSE and REPEAT/UNTIL but no way of forming multi-line procedures or functions in the style of BBC or QL Super BASIC. The special commands that are provided to handle the sound and graphics are fairly easy to use and logical. One serious omission in the Oric 1 has been remedied in the new

". . . new BASIC has not imported the bugs of the old ORIC ROM . . ."

Atmos. The previous ROM lacked a PRINT command that would move the text cursor to any point on the screen so that input positioning proved very difficult. The new ROM solves this problem completely by the provision of a PRINT @ x,y statement which moves the text cursor and hence current printing position to line y column x without changing anything else on the screen. Apart from this the only real additions to the BASIC found on the Oric 1 are the STORE and RECALL commands that can be used to save and load arrays to and from tape.

Perhaps a more important feature of the new Atmos BASIC is that it doesn't seem to have 'imported' any of the bugs in the old Oric ROM. Of course whether there are any newly introduced bugs only time will tell! Certainly all of the old Oric 1 programs that I have tried (around 20) worked without difficulty or evidence of bugs.

Conclusion

The new Atmos is certainly well presented. Its manual is a considerable improvement on the Oric 1 manual and includes a great deal of advanced material that I have spent hours trying to discover for myself when using other machines. For example it gives a list of the machine code subroutines that 'drive' the I/O and a complete list of system and I/O memory locations. For the hardware enthusiast there is even an appendix on how to connect peripherals to the Atmos! This freedom of information is something that other manufacturers would do well to copy.

Oric see the Atmos as the centre of a system that includes an amazing 4 colour printer and a 3" micro disc drive. Calling the Oric printer a printer is something of an understatement as it behaves and functions more like a four colour plotter. As an application system the Atmos plus its two companions is entirely credible but as always the success of any application depends on the availability of the right software.

Finally what hope is there for existing Oric 1 owners now that the Atmos has officially superseded their machine? The answer is "quite a lot"! There seems to be no technical reason why an Oric 1 could not be completely upgraded to an Atmos. The hardware changes would involve simply changing the top of the case for the new keyboard and the software is an even simpler matter of changing ROMs. Whether it is worth changing the keyboard is a matter of opinion (and Oric being willing to supply separate keyboard units) but my advice to all Oric 1 owners is to upgrade to Atmos BASIC. Even if you don't want the extra facilities it offers it is worth it to get rid of the bugs! ■

ATTRIBUTES

b_6	Ø			Ø		
b_5	Ø			Ø		
b_4	Ø			1		
b_3 b_2 b_1 b_0						
Ø	FGND	BLACK	@	BGND	BLACK	P
1		RED	A		RED	Q
2		GREEN	B		GREEN	R
3		YELLOW	C		YELLOW	S
4		BLUE	D		BLUE	T
5		MAGENTA	E		MAGENTA	U
6		CYAN	F		CYAN	V
7		WHITE	G		WHITE	W

Meeting the real world

Joe Pritchard explains how to reconcile a computer with reality in an A to Z of analogue/digital conversion.

An interface is needed to convert the linear, or analogue signals of light, temperature, or physical movement into the digital signals which can be understood by a computer. Such an interface is, sensibly, known as an ADC (analogue to digital converter) or a DAC (Digital to Analogue converter) which performs the reverse process.

A DAC accepts a digital signal from the computer, usually in the form of a binary word, and from its output sends an analogue representation of the signal. The output signal is either a voltage or current, and the magnitude of the signal obviously depends upon the size of the binary input number. Typically the input is in the form of an 8-bit word (the bit length common to most home micros). Some methods of D to A conversion will be examined below, but first a look at the terminology used in the field.

Digital to Analogue

RESOLUTION.

The resolution of a DAC is defined as the smallest increment in the voltage or current output by the DAC that can be produced. This is primarily a function of the number of bits that are in the input word. As an example, let's assume that we have a DAC with an 8-bit input word, and an output of 10 volts for 255 on the input and 0 volts for a 0 input. This is a Voltage Swing of 10 volts, and it can be seen that the smallest voltage change possible at the output will be that caused by changing the least significant bit (LSB) of the input word from 0 to 1. This will be 1/255 of the output voltage swing, giving us a resolution of 0.04 volts. If we have a similar output swing but with only 4-bits at the input, then the change in the output obtained by changing the LSB of the input will be 1/16th of the output voltage swing which gives a resolution of 0.625 volts. Thus the resolution of a DAC increases with the number of bits in the input word. It should be fairly obvious from the above that the 8-bit DAC has 255 separate output voltage values, whereas the 4-bit converter will have only 16 separate output values.

"... each method of conversion has its own advantages and disadvantages ..."

ACCURACY.

The accuracy of a DAC is a measure of the difference between the theoretical output voltage from a DAC for a given input word, and the output voltage that is actually obtained.

LINEARITY.

We might expect that the output voltage obtained for an input word of 200 would be twice that obtained for an input of 100. Occasionally, however, there is some deviation from this ideal, and the circuit is said to exhibit non-linearity.

SETTLING TIME.

This is the time taken between the application of an input word and the output voltage attaining it's final value.

So, armed with some of the most common terms that we will confront, let's examine some methods of performing D to A conversion. It is unlikely that you will ever have to build a DAC from scratch, but an insight into the methods used in conversion using the R-2R method which we will soon discuss. We'll first examine the DAC's that give an output voltage, and then look at those that give an output current.

At the heart of a DAC there is almost always a resistor network of some kind, even in the DAC's that are available as an integrated circuit package.

Another vital component of the DAC is the Summing Amplifier; this is an electronic circuit based upon an operational amplifier. As it is quite important to the rest of the explanation, I shall briefly explain the working of the circuit, which is shown in **Figure 1.**

The output voltage from this circuit, Vo, is proportional to the input voltages, V1, V2, and V3. Thus we can write:

$$V^{OUT} = -K(V^1 + V^2 + V^3)$$

The negative sign is due to the amplifier being connected in a configuration known as an inverting amplifier. The constant K is the gain of the amplifier. If the gain of the circuit is unity then the equation reduces to:

$$V^{OUT} = -(V^1 + V^2 + V^3)$$

This second equation explains why this configuration is called a summing amplifier. So far, we have assumed that all the input resistors are of the same value. If we have a single input resistor, R1, then we can say that:

$$V^{OUT} = \frac{R^4}{R^1} * V^{IN}$$

This can be expanded to give an equation for 3 input resistors. The term R4/R1 gives us the gain of the amplifier in this configuration – ie K in the first equation. Thus we get the below equation:

$$V^{OUT} = -\left[\frac{R^1 *V^1}{R^4} + \frac{R^2 *V^2}{R^4} + \frac{R^3 *V^3}{R^4}\right]$$

Thus it can be seen that by varying the resistors at the input, we can vary the output voltage, even if the input voltage to each resistor is the same. In the binary system, each bit of the binary number has a different significance, and so we should be able to produce an amplifier circuit that has different input resistors for each input, and provide at the inputs a voltage corresponding to a binary 1 or a binary 0. If we wanted the output voltage to depend upon the binary value of the input signals, then

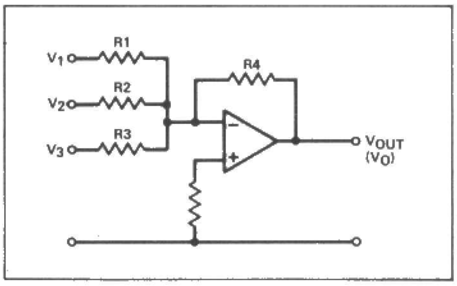


Figure 1. A basic DAC circuit.

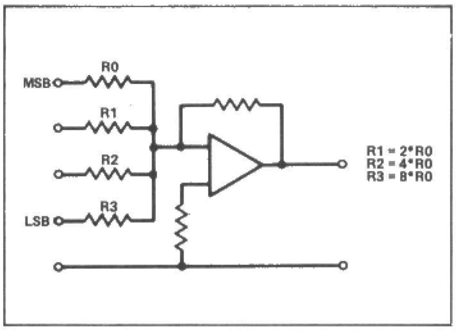


Figure 2. DAC circuit with weighted network.

the value of the resistor that has its input signal from the least significant bit of the binary input should be twice the value of the resistor that gets its input from the next least significant bit of the input word. The actual value of the output will depend upon the resistor R4 in Figure 1. **Figure 2** shows how we could modify the first circuit to give a simple resistive digital to analogue converter. The alteration of the values of the input resistors so as to give a different significance to each input voltage is called *weighting,* and so this arrangement is often called a weighted resistor DAC. Although this circuit is perfectly feasible as a one-off device, there are problems that prevent it from being widely used. The essential requirement is that there is a series of differing resistor values to provide the weighting. They must cover a ratio of 2^{n-1} to 1, where n is the number of bits in the input word.

Thus in the 4-bit DAC shown in Figure 2, the ratio of the highest value resistor used to the lowest will be 8 to 1. Close tolerance resistors would be essential for the input resistors, as if the resistors had, say, a 5% tolerance, we will suffer a variation in the output voltage.

Because of these difficulties, this form of DAC is only really used in low resolution applications. A second form of resistive network DAC is more common, and has been used on many commercially available DAC chips. This is called the R-2R DAC.

The primary advantage of the R-2R device is that it only uses 2 different values of resistor in the conversion network, thus making it easier to produce such a network as a component in an integrated circuit. The diagram of a 4-bit R-2R DAC is shown in **Figure 3.** The other thing to note is that more resistors are needed for this arrangement.

One thing that you should have noticed is that the circuits so far discussed accept their inputs from switches. In real DAC integrated circuits, these switches are of an electronic nature: common switches are FET's, Bipolar transistors or switches fabricated using CMOS technology. The digital signals applied to the inputs of these DACs thus control the switch, and so supply a voltage to the input resistor concerned. The use of switches in this way means that voltages can be applied to the input resistors that are different to those normally available in the logic circuitry. Thus, it now becomes easy to vary the output from a DAC by changing this voltage that is switched by the electronic switches. This voltage is called the *reference voltage,* and is fixed in some devices but is user variable in others. An example of a chip in which it is fixed is the ZN428, and in the ZN425E the user may supply his own reference voltage or may use that on the chip. A common reference voltage on chips for an 8-bit DAC is 2.5 volts. The ZN425E is probably one of the most widely used devices in this field, and this uses an R-2R conversion network.

Some DACs produce a current as output rather than a voltage, one such is the Motorola 1408L8. These devices are not as common as the others we have considered but they work on similar principles.

The input binary code switches current instead of voltage, and these currents are provided by transistors and a resistor network of some sort. The currents are then added together by an operational amplifier circuit, and the output current delivered to the outside world. Some devices convert the current thus generated into an output voltage, although this is an easy enough job to do off the chip. Current can be converted to a voltage by a simple resistor or an operational amplifier circuit.

Commercial DACs are available in a wide range of settling times and bits of conversion. Devices that have between 6 and 12 input bits are common, and the settling times of DACs range between 25ns and 0.1ns. The latter figure is for a high precision DAC.

We shall now go on to examine Analogue to Digital converters. Again, there are a few technical terms to consider here before we actually consider the techniques.

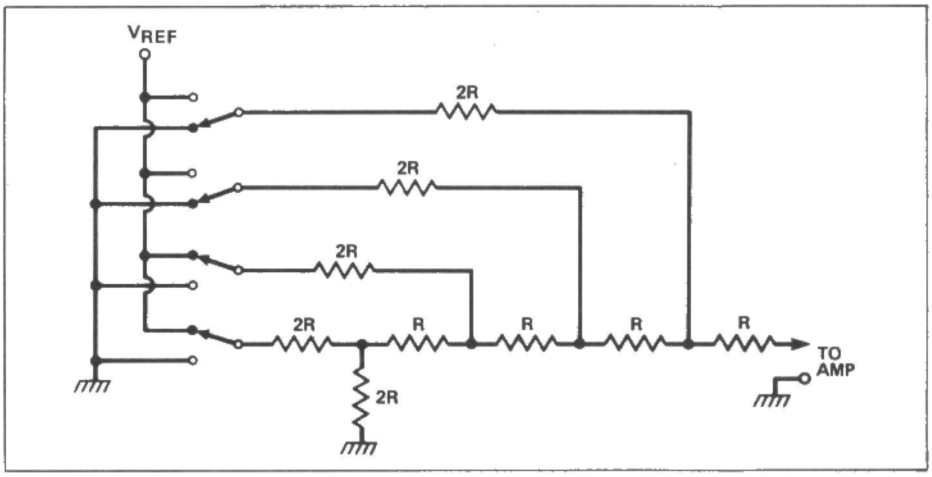


Figure 3. Circuit of a simple R-2R DAC.

Analogue to Digital

QUANTIZATION ERROR.
This is an error in the process that is caused by the fact that only a limited number of binary output codes from an ADC must codes for an infinite number of slightly different input voltages. Thus, the voltages 0.0002 and 0.0006 volts when applied to an ADC could very well give the same digital code at the output of the ADC.

INPUT VOLTAGE.
This is the range of voltages that may safely be applied to the input of the ADC. It can be unipolar, in which case one of the limits is zero – ie giving us 0 to 5 volts or 0 to —5 volts – or can be bipolar, in which case one limit is a negative voltage and the other limit is a positive voltage. Thus a bipolar input voltage range might be —5 to +5 volts.

CONVERSION TIME.
This is fairly obvious. It is the time that the converter needs to produce at it's outputs a binary code corresponding to the input voltage that is applied. This can be between 1 and 100 microseconds, although there are some converters that are slower than this and some that are faster.

Analogue to Digital converters usually have one input to which a voltage is applied, and the output is provided on a series of output lines that carry a binary representation of the voltage applied to the input. The output is thus of a parallel nature, and the signals at the output are usually compatible with TTL or CMOS logic systems. There are various methods by which the voltage can be converted to a digital signal, and when used in conjunction with a computer, they each have different requirements in terms of the software needed to read the digital signals. For some of these methods, the software overheads, as they are called, are low but in some methods they are quite high and the speed of conversion can be limited by the speed at which the computer can control the process of analogue to digital conversion.

A fundamental component of ADC's is the circuit element called the *comparator.* This is a device that can indicate which of two input voltages is the highest. This is all we really need to know about this device, but anyone interested can find details in most electronics textbooks. The symbol for this element is shown in **Figure 4.**

We will now go on to discuss the three most common methods of conversion of analogue signals into digital signals. These are the techniques of *successive approximation, slope conversion,* and *voltage to frequency conversion.* Of these, the latter has the highest software overheads as we shall soon see.

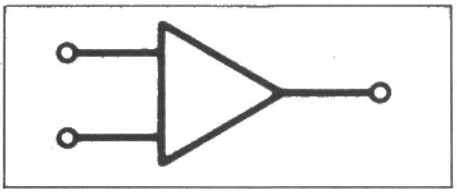


Figure 4. A comparator's circuit symbol.

SLOPE CONVERSION.
There are some alternative names for this form of conversion, such as *ramp conversion* or *counter controlled conversion.*

There are three main components to a simple counter controlled conversion system. These are a counter, a comparator and a digital to analogue converter of some sort. The counter must be such that it has as many output terminals as there are to be bits of input to the computer. A block diagram of such a system is shown in **Figure 5.** Obviously, in the real world all these components are to be found on an integrated circuit. Analogue to digital conversion based upon this technique is performed by the ZN425E chip with the addition of just a few more components.

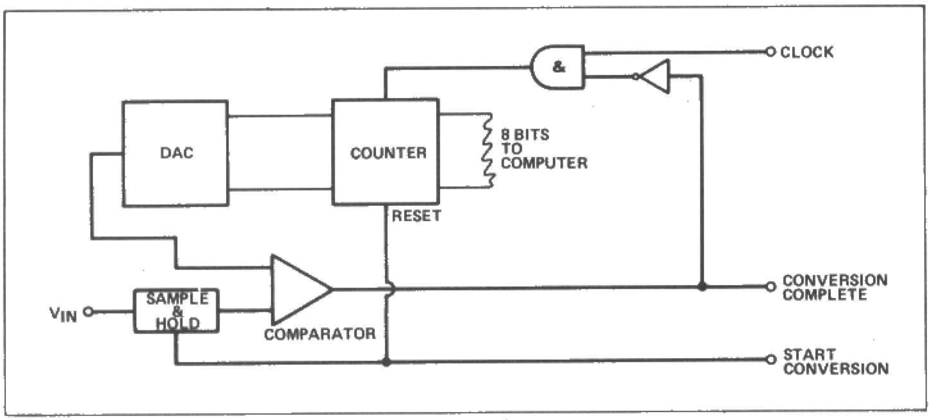


Figure 5. Basic blocks of a counter controlled converter.

The operation of the circuit is quite straightforward. A clock signal consisting of a square wave is supplied from either the computer or an external clock circuit, and this causes the 8-bit counter (assuming we're considering an 8-bit ADC) to increment. The signal at the output of the counter is made available to both a DAC and the data bus of the microcomputer being used. The voltage generated by the DAC is applied to the comparator, where it is compared to the analogue voltage from the sample-hold circuit. This circuit is connected so as to "remember" an analogue voltage that was present as it's input at a given instant in time. Obviously, this circuit would be triggered by the computer sending a pulse down the "Start Conversion" line. As soon as the voltage generated by the DAC exceeds that from the sample-hold circuit, the output of the comparator goes high, signalling to the computer via the "Conversion Complete" line that the process is complete and that the value held on the counter is a digital representation of the analogue voltage applied. As soon as the conversion complete signal is sent, the AND gate on the clock line prevents further clock pulses from incrementing the counter. The computer can then reset the counter and read the next analogue voltage into the sample-hold circuit by pulsing the "Start Conversion" line. The speed of conversion of this type of ADC is dependant upon the value of the analogue signal that is under consideration and the rate at which the counter is clocked.

If a low clock rate is present, then the counter will increment more slowly, and so will take longer to reach the value needed to switch the comparator for a given voltage than would an ADC of this type with a faster clock rate. We can calculate the time it takes to achieve a count of 255 on the output of the ADC for a given clock rate, by dividing 255 by the clock rate in hertz. The other consideration with regards to conversion time is the magnitude of the voltage applied. Obviously, for a given clock rate, a lower voltage requires a smaller count on the counter to occur (before the comparator switches) than does a higher voltage. The longest time that will be encountered because of this limitation is that time needed to achieve a count of 255 at the output, as we have already considered above.

SUCCESSIVE APPROXIMATION.

This has higher software overheads than the first method that we considered, but it's conversion time is irrespective of the magnitude of the voltage applied to the input of the sample-hold circuit. The hardware requirement is shown in **Figure 6.**

The mode of operation of this circuit is as follows. When a conversion is required, the computer first pulses the start line to activate the sample-hold circuit. The port is then set to zero, and program in the computer turns on the most significant bit of the port. This will generate a voltage at the output of the DAC, and the computer will be able to tell if this voltage is greater or smaller than the input voltage by the state of the output of the comparator. If the voltage thus obtained is greater than the input voltage, then the bit is set back to zero. If it is less than the input then the bit is left at one. The same process is then repeated for each of the other bits thus building up an 8 bit binary number representing the input voltage. This method will take as many steps as there are bits in the required result. The software will have a copy of the final value output to the port and this value will be therefore available to be read by other programs. The software will be responsible for knowing when the conversion is finished, a job that was done to a greater extent by the hardware in the Slope Conversion method. The successive approximation method is much faster than the first method that we considered, and the limitations to speed are set by the settling time of the DAC and the speed of the computer software.

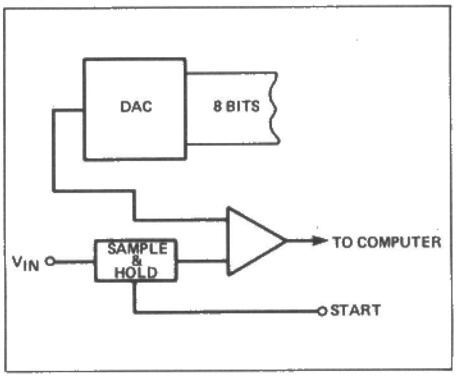


Figure 6. SAR converter.

VOLTAGE TO FREQUENCY CONVERSION.

This method of analogue to digital conversion has low hardware costs but a large software overhead. It is different from the other methods in a couple of important ways. In the methods previously discussed, there was a definite point at which conversion was over, and at this point a binary number representing the analogue value was made directly available to the computer. In this method, there is no fixed end point to the conversion process, and the computer has to form a binary number from the output of the ADC, which is a stream of pulses. The frequency of this square wave is dependant upon the voltage applied to the input of the ADC at that moment in time. By measuring this frequency using software, the computer can obtain an indication of the voltage being applied to the ADC. **Figure 7** shows an example of the sort of circuit that can be used. It is essentially a voltage controlled oscillator.

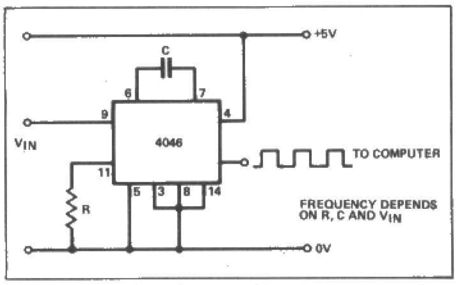


Figure 7. Basic voltage to frequency converter.

There are many circuits that would serve in this role, and an obvious candidate would be the 555 timer integrated circuit. The computer then has to calculate the frequency of the incoming signal and then convert the frequency measured into a voltage reading. If the micro concerned has an internal real time clock, then the task of measuring frequency is fairly easy. The circuit has to be adjusted so as to generate a square wave of fixed frequency at a given voltage. This is not all that easy, and in practical circuits based upon this principle considerable variation of the input frequency can occur unless care is taken.

In this article, I've looked at the most commonly used techniques of A to D and D to A conversion. There are, however, other methods of conversion that have not been looked at here, each with their own advantages and disadvantages. ■

SOFTWARE REVIEWS

Adam Denning reviews a selection of the latest utility software.

Devpac

Spectrum 48K
Hisoft £14.00

Devpac is, no matter what other software authors may say (see letters), the ultimate machine code development system for the Spectrum owner. At £14.00 it provides more facilities and works better than any other on the market.

The package comprises a cassette containing the editor/assembler and the monitor/disassembler, plus a thick manual describing every command in detail. The manual also provides at least one example for getting used to both the assembler and the monitor, and is an essential read for all users.

The assembler, known as GENS, is based very closely on the Zilog Z80 original design, but naturally it takes advantage of some of the Spectrum's features too.

The first notable thing about GENS is that it is totally relocatable. This means that it can be loaded into any convenient part of memory, a feature which no other Spectrum assembler can offer. The program is loaded by typing LOAD"" xxxxx, where xxxxx is the address to which you want the assembler to be loaded. A simple RANDOMIZE USR xxxxx gets the program started, and the user is prompted to give the buffer size; this is the size of the buffer that will be allocated for the Include directive (see later), and can vary from 0 to 9. As with all GENS commands, this can be defaulted simply by pressing ENTER.

Having done this, the integral editor is initialised, and a prompt with a flashing cursor is produced. The editor is an extremely comprehensive line editor, and so many and diverse are its functions that the beginner is certain to be confused. This is where the excellent manual comes in. An hour or two spent with this will soon clear up any problems.

Examples of the commands available within the editor are:

I – this can be followed by up to two parameters, the start line and increment, and causes the editor to produce line numbers automatically.

N – this must be followed by two further numbers, and renumbers the entire file contents.

F – will find a string between specified or defaulted line numbers. After one use of the F command, the search string can be defaulted.

S – this does the same as F but also replaces the search string with another.

D – deletes lines between specified limits.

There are of course far more facilities than this, including excellent editing features that even show up Spectrum BASIC's editor.

Having written one or more source files, which may be stored in memory, on cassette or on microdrive, one can then assemble it (them). GENS has all the usual ORG, DEFGB, DEFW, DEFS and DEFM assembler directives, and can optionally produce listings to both screen and printer, a cross-referencing symbol table, checked object code or un-checked object code.

'. . . it is impossible to recommend any other development package for the Spectrum over Devpac'

Whenever a *F is met in the source file, GENS opens a file (from either tape or microdrive) and looks for another source file in the correct format. Once found, this file will be loaded in a bufferful at a time, and *included* in the assembly as though it were part of the source file resident in memory. As many of these include files as you wish can be used in an assembly, but of course you must take care to allocate enough symbol table space for each one. Using this facility, it is an easy matter to assemble say 100K of source into the corresponding machine code.

What more can one say of an assembler? It certainly does its basic job as well as any other, but its added extras put it on a higher level than the rest. Excellent.

This takes us on to the monitor and disassembler, called MONS. Like GENS it is again totally relocatable, and considering its power it is also very small. Upon running, it presents the standard Zilog 'front panel', with all the registers and their contents being displayed, and the memory adjacent to the memory pointer being shown too. The instruction currently pointed to by this memory pointer is shown disassembled at the top, as is the address. All numbers default to hex, but can be switched to decimal at the press of a key. From here on in, forget that it is only a Spectrum below your fingertips. MONS gives you the ability to execute a machine code program one instruction at a time. This is known as single-stepping, and is a pre-requisite on a large machine but is rare on a micro. One very unusual facility that MONS offers in this department though is the ability to single-step through ROM as well as RAM. Breakpoints are dynamic, being stored just above MONS in memory, and so as many breakpoints as memory can provide can be set. You could for instance set up breakpoints after every subroutine in code, which may be useful if you know your program doesn't work but don't know where.

Whenever a breakpoint is reached, MONS is restarted, with the familiar front panel being displayed again, and all register and memory contents left just as they were when the breakpoint was reached. It is this sort of thing that makes assembly language programming a lot less awkward.

Of course the ability to single step is all very well, but what if you know that some routines work but others don't, and the ones that don't call the ones that do? Well, MONS caters for these too; during single step, one can choose whether or not to follow calls, jumps and returns. One can also set breakpoints after the next CALL instruction, jump to the routine in question and execute it in real time, and then return to the monitor at the next instruction.

Other facilities are hex string search, memory adjust, block move and block fill, and of course disassembly, among others. Disassembly can be invoked either by pressing Symbol Shift-$, when the memory from the memory pointer is disassembled 20 instructions at a time, or by use of the 'T' command, which is rather more powerful.

With such power at such a low price, it is impossible to recommend any other development package for the Spectrum over Devpac. Others do the job, but none so elegantly.

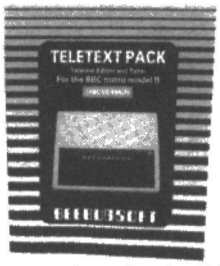
Teletext

BBC Micro B
Beebugsoft £10 (cassette)
£12 (Disk)

Teletext is another programming aid for creating Mode 7 screens on the BBC. Our version was supplied on disc, and is booted by pressing SHIFT/BREAK in the normal way. This presents a title page with the Beebug logo, which disappears after a few seconds and is replaced by a menu.

The menu provides two options: selection of the Teletext Tutor, or selection of the Editor itself. The tutor is intended to teach the ins and outs of Teletext (Mode 7) display creation – it fails miserably. The potential for a good tutor here is enormous, as Mode 7 graphics ARE difficult to get to grips with at first. BeebugSoft seem to have commissioned entirely the wrong person to write it. It is written in the most unbelievably unfriendly manner, and assumes far too much knowledge of the BBC and its BASIC.

For those who are not conversant with teletext graphics, the BBC User Guide together with the manual supplied with this program will prove far more informative. A pity.



However, the editor itself is quite a different kettle of fish. One thing that must be cleared up now is that there is quite a difference between teletext editors intended for a Prestel type system (such as the one featured in *E&CM* this month) and those editors intended for more general applications. Prestel has its own set of clearly defined but somewhat idiosyncratic standards, which other editors simply do not need to follow. The BeebugSoft one falls into this latter category, and is certainly a worthy piece of code in this respect.

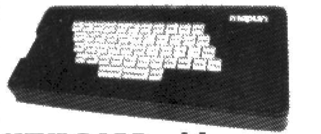
All the teletext standard codes and commands can be used in the easiest of ways – simply be pressing a function key. A keyslip is also provided to fit over the red function keys, making the remembering of all functions that much simpler.

Invocation of the editor results in a menu being displayed, giving a choice of options from quitting the editor to saving a screen. Choosing the 'edit a screen' option then presents the user with a blank screen if the program is newly invoked, or the results of previous attempts if not. On this screen is a flashing cursor, which can be moved around in the normal fashion, and characters can be entered at the current cursor position. Like all good editors, double-height dual insertion is also taken care of.

If satisfied with the thus created screen, the user can save it to disc/tape in the normal *SAVE filename 7C00 7FFF fashion, or more interestingly a BASIC procedure which can then be used in other programs.

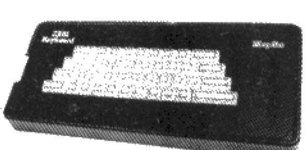
In all, this package provides Teletext editing facilities in a well thought-out manner, but fails to give the novice that essential degree of help.

EXCITING ADDITIONS FOR YOUR HOME COMPUTER



KEYBOARD with ELECTRONICS for ZX SPECTRUM

★ Full size, full travel keyboard that simply plugs into expansion port on your Spectrum. ★ Offers single key selection of all major multi-key functions. ★ Extends port for other peripherals. ★ Can accept Atari-type joysticks (optional extra — order 2 of FG66W, £1.36 each and note that case will require cutting).
Three kits needed to build unit: Order LK29G, LK30H & XG35Q. Total price £39.95. Full construction details in Project Book 9 XA09K 70p. Also available ready-built. Order As XG36P. Price £44.95.



KEYBOARD with ELECTRONICS for ZX81

★ Full size, full travel keyboard that's easy to add to your ZX81. ★ No soldering in ZX81; simple instructions make it easy to fit. ★ Makes Shift Lock, Function & Graphics 2 single key selections.
Complete kit (excl. case) LW72P Price £23.95. Case XG17T £4.95. Full construction details in Project Book 3 XA03D. Price 70p. Ready-built in case XG22Y. Price £32.50.

MODEM

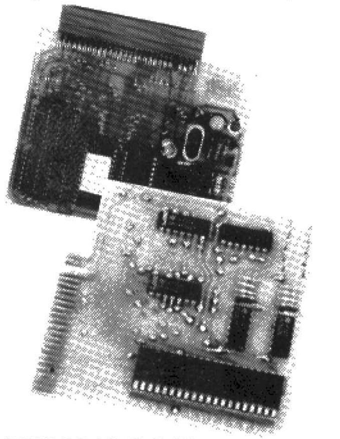
A CCITT standard modem that connects directly to your telephone line via a BT approved transformer. Transmits and receives simultaneously on European standard frequencies at 300 baud. May be used to talk to any other 300 baud European standard modem including the Maplin Computer Shopping modem on 0702 552941 and any British Telecom Datel 200/300 Service modem. The modem's computer interface is RS232 compatible.
Complete kit (excl. case) LW99H. Price £44.95. Case YK62S £9.95. Full construction details in Project Book 5 XA05F Price 70p.

INTERFACES for MODEM

Interfaces are now available for the following machines: Commodore 64, Dragon 32, Oric, Spectrum, VIC20 and ZX81. Each is complete with a Machine Code Communications program. The BBC micro needs no interface and a suitable program is on Maplin catalogue page 15 or Project Book 8, page 59.

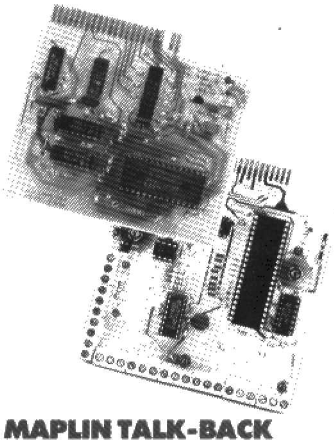
Computer	Order	Details	Price
64/VIC20	LK11M	Book 7	£9.45
Dragon 32	LK12N	Book 8	£13.75
Oric 1	LK40T	Book 10	£12.95
Spectrum	LK21X	Book 8	£17.95
ZX81	LK08J	Book 7	£24.95

**Project Book 7 XA07H. Price 70p.
Project Book 8 XA08J. Price 70p.
Project Book 10. XA10L. Price 70p.**



ZX81 I/O PORT

★ Provides two bi-directional ports for 16 input or output lines. ★ One buffered output which can interface directly to CMOS. ★ On board address selection permits expansion to six ports with two boards.
Complete kit LW76H. Price £9.25. Full construction details in Project Book 4 XA04E. Price 70p.



MAPLIN TALK-BACK SPEECH SYNTHESISERS

★ Unlimited vocabulary with allophone (extended phoneme) system. ★ Can be used with unexpanded Oric 1, VIC20 or ZX81 as it does not require large areas of memory. ★ Speech may be easily added to programs. ★In VIC20 version speech output is direct to TV speaker with no additional amplification needed.

Computer	Order	Details	Price
Oric 1	LK28F	Book 9	£22.95
VIC20	LK00A	Book 6	£22.95
ZX81	LK01B	Book 6	£16.95

**Project Book 6 XA06G. Price 70p.
Project Book 9 XA09K. Price 70p.**

DRAGON 32 I/O PORT

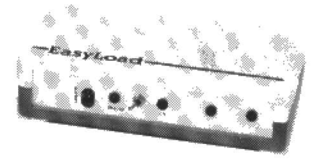
★ Provides two TTL & 3-state bus compatible 8-bit ports, ★ Four norm/inv. latched ports, ★ Two relay switched ports ★ And two opto switched ports. ★ Module plugs directly into cartridge socket and is fully programmable from BASIC.
Complete kit LK18U. Price £13.95. Full construction details in Project Book 8 XA08J. Price 70p.

MAPLIN CATALOGUE

Full details of all Maplin's projects and electronic components in our huge 480 page catalogue. On sale now in all branches of W.H. Smith price £1.35. Or send £1.65 (incl. p&p) to our mail order address.

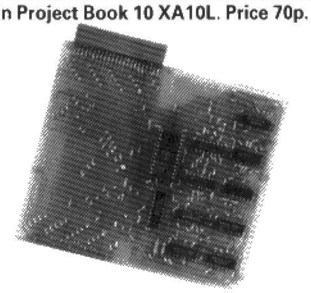
OTHER PROJECTS

For full details of our other computer-related projects please see the relevant project book as below:
**ZX81 Sound on TV — Book 6.
ZX81 Extendi-RAM — Book 9.
VIC20 Extendiboard — Book 9.
Dragon 32 Extendiport — Book 10.
TTL/RS232 Interface — Book 9.
Project Book 6 XA06G. Price 70p.
Project Book 9 XA09K. Price 70p.
Project Book 10 XA10L. Price 70p.**



SPECTRUM EASYLOAD

★ Greatly reduces cassette LOADing & SAVEing problems on Spectrum. ★ Battery powered, no bus connections. ★ Charging from Spectrum PSU. ★ SAVE & LOAD indicators.
Complete kit (excl case) LK39N. Price £9.95. Full construction details in Project Book 10 XA10L. Price 70p.

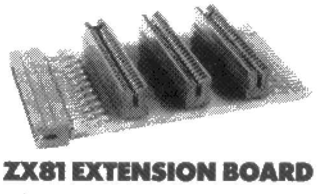


ZX81 HI-RES GRAPHICS

★ Full 256 x 192 fine pixel display with normal or inverted video. ★ Draws lines, circles and triangles, fills and textures. ★ Up to 32 user defined graphics. ★ Operates directly from extended BASIC.
Complete kit LK23A. Price £19.95. Full construction details in Project Book 9 XA09K. Price 70p.

ZX81 SOUNDS GENERATOR

★ Turns your ZX81 into a mini-synthesiser. ★ 3 programmable tone generators. ★ 3 programmable attenuators. ★ Noise generator with 3 pitch levels for special effect sounds. ★ Single address access with PEEK & POKE. ★ Connects directly to expansion port socket or extension.
Complete kit LW96E. Price £10.95. Full construction details in Project Book 5 XA05F. Price 70p.



ZX81 EXTENSION BOARD

★Plugs directly into ZX81 expansion port. ★ Accepts a 16K RAM pack and three other plug-in modules simultaneously. Parts are sold separately as follows:
PCB GB08J. Price £2.40. Edge Connectors (4 needed) RK35Q. Price £1.96 each. Track pins (1 pack needed) FL82D. Price 85p per pack of 50.

MAPLIN
ELECTRONIC SUPPLIES LTD

Maplin Electronic Supplies Ltd. Mail Order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Tel: Southend (0702) 552911. • Shops at: 159-161 King Street, Hammersmith, London W6. Tel: 01-748- 0926.
• 8 Oxford Road, Manchester. Tel: 061-236-0281. • Lynton Square, Perry Bar, Birmingham. Tel: 021-356-7292.
• 282-284 London Road, Westcliff-on-Sea, Essex. Tel: 0702 554000. • 46-48 Bevois Valley Road, Southampton.
Tel:0703 25831. All shops closed all day Monday.
All prices include VAT and carriage. Please add 50p handling charge to orders under £5 total (except catalogue).

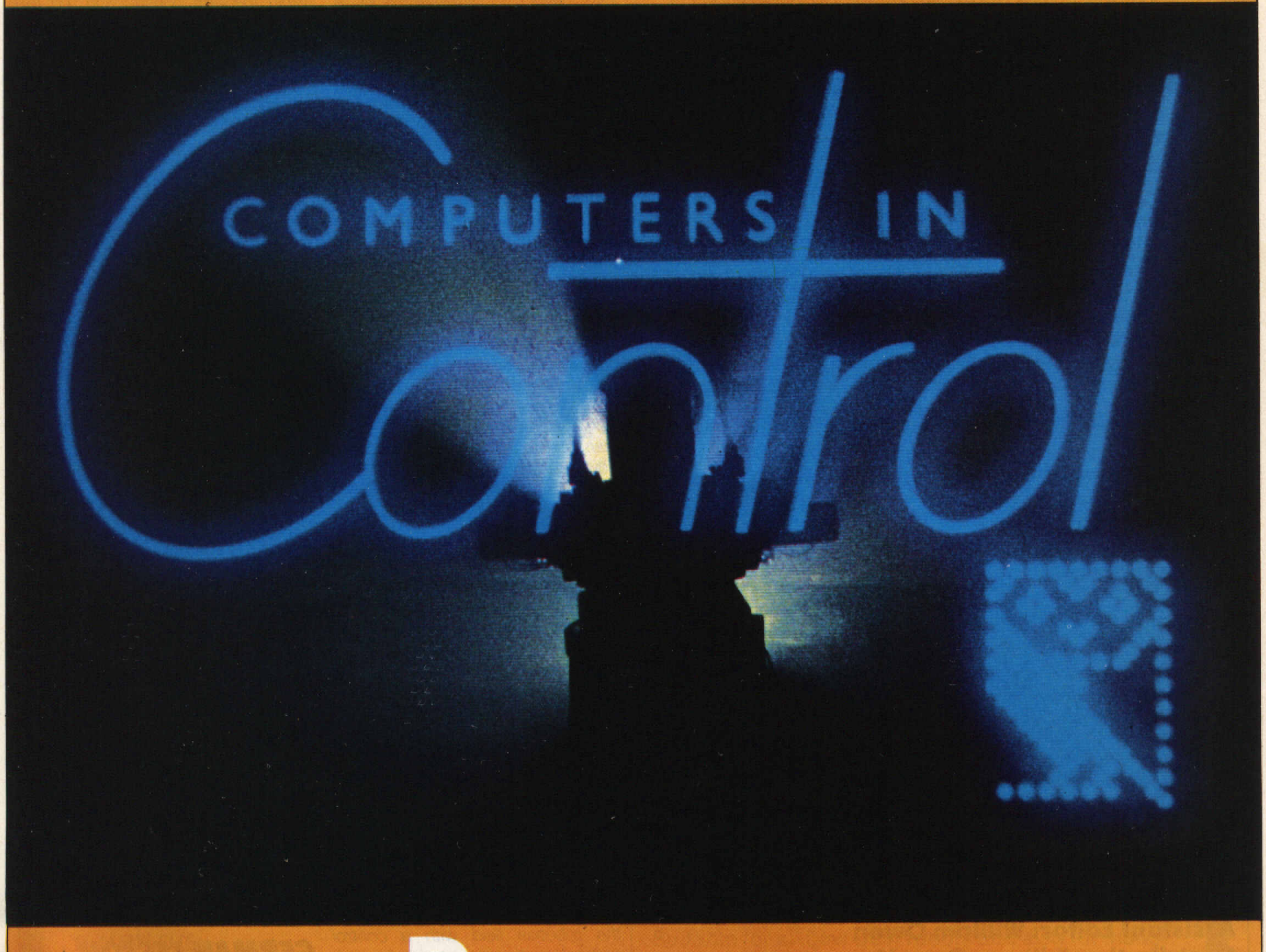
FREE

Your ROBOT

An EMAP Publication

BRITAINS FIRST ROBOTICS MAGAZINE APRIL 1984

THE 'ROBOT PROGRAMME'



PREVIEWED

▪ HERE'S LOOKING AT YOU ▪

ROBOT VISION SYSTEMS ▪ SOLID STATE TV CAMERA PROJECT

PLUS: ROBOT CONTROL LANGUAGES

EDITORIAL

VOLUME 1 ISSUE 3

The team responsible for BBC TV's Computer Programme have for the past few months been working on a new series – Computers in control, or as we have dubbed the series 'The Robot Programme'. The fact that the BBC have decided to dedicate valuable air time to an exploration of control systems, from basic servo controllers to sophisticated designs featuring a number of interactive processors, is a sure sign of the importance attached to an understanding of such systems.

In this issue of **Your Robot** we provide full details of the programmes, contents and timings although, in typical BBC fashion, we have been asked to point out that these may be subject to variation. We are sure that the series will be of interest to readers of **Your Robot** and, even though some of the programmes may have been transmitted by the time you read this, a healthy number of repeats should ensure that you can catch the whole series.

LOW COST VISION

Experimenting with vision systems has to date been rather a costly affair, involving the purchase of a vidicon camera costing in excess of £150 and some form of A/D converter, before any work could begin. This month we describe a system that is compatible with the Spectrum and costs around £5, yes five pounds, to build.

The solid state camera is based on a low cost DRAM and, as is the case with most aspects of robot building, although a degree of manual dexterity is required, the project can be tackled by anyone capable of handling a soldering iron.

We are particularly keen to hear from readers who build the robot vision system. Details of applications will, we are sure, be of interest to many of our readers.

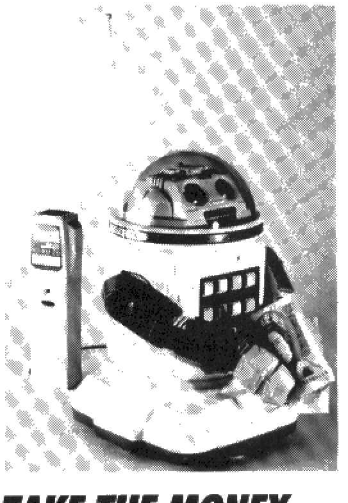
CONTENTS

Cover picture courtesy BBC TV

Editor: Gary Evans
Assistant Editor: William Owen
Contributing Editors: Gary Herman,
Peter Matthews, Richard Moyle
Telephone: 01 833 0846
Advertising Manager: Clare Fullerton
Executive: Richard Jansz
Classified: Andy King
Telephone: 01 833 0531

YOUR ROBOT
2nd FLOOR
155 FARRINGDON ROAD LONDON EC1R 3AD

NEWS



TAKE THE MONEY AND RUN

Only thirty five times that greenback clasped between its grippers will buy the Tomy voice recognition robot. A remarkable machine with a remarkable price; it is capable of obeying eight instructions from his master's voice, has a limited vocabulary, and operates under radio control over a range of 5 metres.

The sophisticated voice recognition system understands the commands stop; talk to me; pick up; put down; go forward; go back; turn left; and turn right. The two arms have two axes of movement (at the shoulder and wrist).

This 20th century version of man's best friend is manufactured by a Japanese toy company which is better known for its mobile motor-driven Star Wars type beasts called Zoids, whose perambulatory skills would put the efforts of many a professional robotics engineer to shame. The voice recognition robot was the logical next step, and has brought technology down to a realistic price. Perhaps computer control for under £50 comes next?

YOUNG TRAINER

Northern Computers have entered the robotics field with a control technology training course for children – Youngtrainer.

The device hooks up to the BBC or Vic-20 and can simulate traffic lights, house lights, a burglar alarm, washing machine under software control. The range of software and overlays can be extended to simulate many control mechanisms. A further development by Northern Computers, a robot arm, will bring real life control into primary schools.

The Micro Pulse Youngtrainer costs £80 and is available from Northern computers, Churchfield Road, Frodsham, Cheshire.

GRIPPING OGRE

L. W. Staines & Co. is a Barking based form which, using its experience in computer controlled machine tooling has built a relatively strong and powerful robot arm for the home, educational and light industrial market.

The Ogre is priced at £195, well below that of many less capable rivals. It uses Staines' own gearboxes, capable in normal circumstances of lifting 20lbs., and within the arm construction, up to 2lbs. Ogre is powered by a single DC motor with an accurate optical counter.

Ogre has 5 axes of freedom and a powered gripper. It is an expandable system, with the option of adding an extra arm, travelling base unit or special purpose gripper. The Ogre easily interfaces with any home micro with an 8-bit I/O port. More on the machine next month when **Your Robot** will give a full review.

ARMDROID FOR INDUSTRY

Colne Robotics have released specifications for their latest robot arm, the Armdroid II, which the company claims will improve on the performance of Unimation's Puma range at one tenth of the price.

Each of the seven joints of Armdroid II is equipped with a Z80 processor, and there is an on-board Z80 supervisor board, closed loop control using optical encoder feedback, and sophisticated on-board EPROM software.

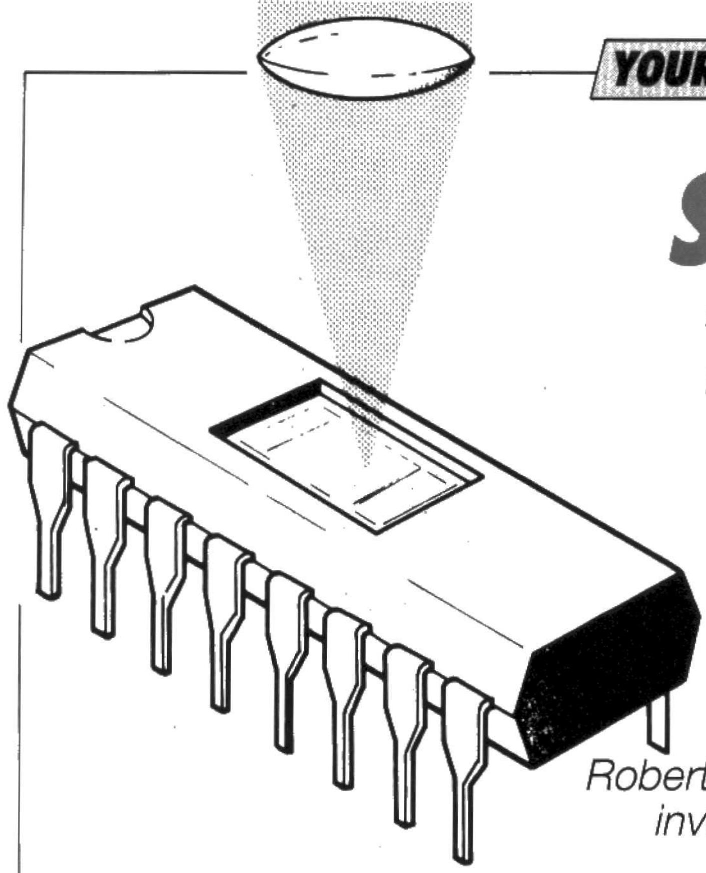
The machine has seven degrees of freedom, a load capacity of 2kg, and a range of grippers including a two finger variety. Armdroid II can communicate with other computers, and operates under simple commands. There is a point-to-point 'learn by the nose' learning facility.

Colne say the new machine will be very fast, and expect it to extend their market from education and research into light industrial assembly. Armdroid II will be priced at around £2000.

GERMAN LESSON

A robot is to join a select group of UK manufacturers in presenting British control technology to key German educationalists in Dusseldorf next month.

The Economatics BBC Buggy is one of a range of British-made computer equipment which will be exhibited in support of a seminar presented by five major British educational experts at the German Engineers Association – Verein Deutscher Ingenieure – from February 13th-15th.



SOLID STATE TV CAMERA

Industry-standard DRAM chips can be made into simple "eyes" which may not rival the Vidicon camera, but which are certainly a cheap way of experimenting with tone and pattern recognition. Robert Harvey and Richard Sargent have been investigating this visual aspect of Robotics.

The mechanical functions of the human body have proved relatively easy to duplicate and robotic devices may now be given a sophistication of movement which can be both more powerful and more accurate than that possessed by mere mortals. The microprocessors lurking behing the valves and solenoids are already performing highly skilled tasks and as every year passes they become slightly more powerful and it is naturally quite tempting to begin to think of them in terms of the "brain" of the machine. However, if robots are to be used outside the somewhat specialised environments of, for example, the car-production assembly line or the surface of Mars, then they will have to be given an intelligence which is several orders of magnitude higher than that which they already have.

In short, if robots are to work safely in close proximity to human beings then they will need to be given the senses of sight, sound (hearing and speaking), touch and, to a lesser degree, smell. Behind these senses there would need to be a collection of programs interpreting and reacting to the received data at a speed far beyond that which the present generation of microchips are capable of producing. Sight is arguably the most important of the senses and the one in which the home constructor can experiment with the absolute minimum of hardware costs.

IMAGE SENSING

The circuitry and software outlined in this article represent a modest yet worthwhile taste of what image sensing is all about. Work in this field forms a very important part of the general work on Artificial Intelligence (AI) and to say that there is a gap between the hardware development (the "eyes") and the software development (the "brain") is something of an understatement. If we give a robot the ability to see, does it see the sort of image we want it to see? Is it capable of interpreting that image at all, and will it interpret the image in a way that suits us? Can it learn from its visual experiences? These questions and other like them are being tackled by researchers in the field of robotics and AI.

THE HARDWARE

The hardware needed to convert light into electrical signals has been around for quite a few decades. The standard video camera is a fairly high-resolution device, but it is not exactly fast, and it deals in analogue rather than digital information. The more recent PHOTODIODE is the best sensor available and is already used widely for industrial control purposes. An array of photodiodes on a chip of silicon can provide a matrix which is similar to the cone and rod layer of the human eye, except that it will not interpret colour. The chip may be put in the film plane of a camera, and with associated logic circuitry it forms the small and compact Solid State TV Camera currently available from manufacturers today. Readers interested in the circuitry and construction of such cameras will find the article by A. Longford and E. Reticon on page 17 of the September 1981 edition of *E&CM* most illuminating, if you will forgive the pun.

THE D-I-Y VERSION

The high density dynamic RAM chips currently available consist of discreet P-silicon/N-silicon junctions laid out in arrays similar to that of the photodiode-matrix chip. The junctions act as capacitors which hold their charge for a limited period. In the case of the DRAM chip the charge on the capacitor represents one BIT of information – a logical low or high. If the chip is not regularly REFRESHED, those capacitors that have been charged by a WRITE operation lose their charge, the memory content of the entire chip becomes invalid and you have an irretrievable memory crash.

LET THERE BE LIGHT . . .

The rate at which the capacitors discharge is proportional to the amount of light falling on a particular P-N junction. This is normally of no significance whatsoever as the chip surface is encapsulated in an opaque medium. Uncover the chip surface and you obtain an array of light sensitive cells – a sort of primeval image sensor.

The experiment on image sensing can therefore take place with a computer, an I/O port and a 4116 DRAM chip. (A 4164 DRAM should be used later, after some experience has been gained using the 4116 chip). The circuit could hardly be more straightforward. It consists of a PIO chip which directly controls the individual memory cells of the 4116 as shown in **Figure 1.** All the hard work is done by the software and a Spectrum proved handy for this particular task. Constructors who scrutinise the code with a view to changing it to run on a different Z80 machine ought to be aware of the fact that the Spectrum has a CPU clock of 3.5MHz, but that this value is not constant (the Spectrum ULA controls the CPU clock) and that the bit-mapping of the Spectrum high resolution screen is somewhat peculiar. Owners of 16K Spectrums should also be aware of the fact that machine code will run marginally more slowly in RAM beneath 8000H, since this RAM is partly video RAM and also controlled by the ULA.

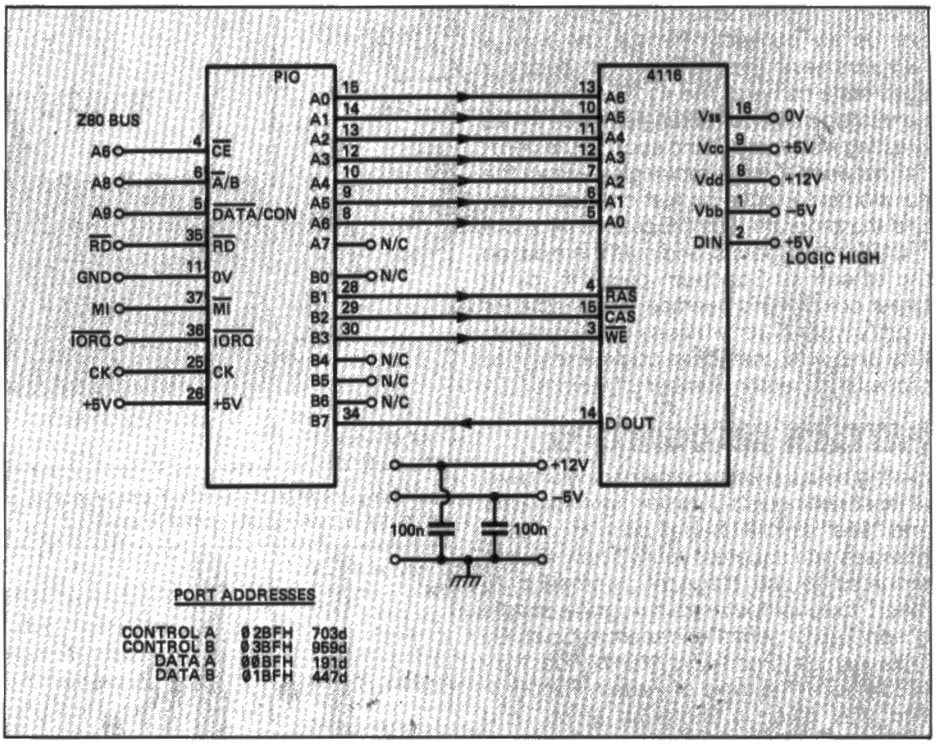
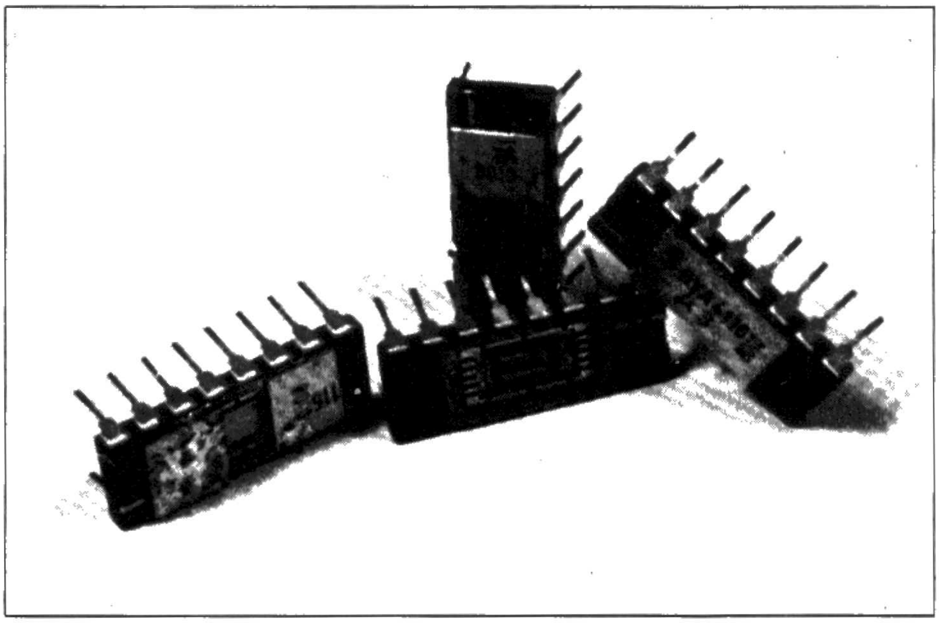


Figure 1. The hardware of the Solid State TV camera consists simply of the DRAM image sensor itself and a Z80 PIO.



Suitable DRAMs can be identified by their packaging and the metal plate, once removed, reveals the capacitor array. The author's initial attempts at chip surgery proved less than successful.

Taking the tops off 4116 DRAM chips proved to be quite simple, though you should be prepared to "write off" the first one you mutilate. There are a number of things which can go wrong (and did go wrong in the case of our prototypes). However, if you have a steady hand and follow the instructions, then all should be well.

The design of chip which should be used is the type known as Ceramic Dual-in-line Welded Seal. The pins are side braxed and gold-plated, which although not important, makes the chips easy to recognise. The body of the IC is usually grey in colour and a metal plate is soldered or welded across three-quarters of the top surface. The ICs used in the prototype bore the code HYA4116 A3. The metal plate has to be removed. The first attempt involved using a fine-toothed saw, with the IC already soldered on a piece of veroboard to keep it rigid. This worked, but put fine swarf and foreign bodies onto the surface of the chip which then had to be blown off. In fact the solder/weld line is so thin that the plate can be cut and prised off using a Stanley-knife and this method produces hardly any loose material. Heating the plate with a soldering iron until it slips off also proved successful. On the exposed chip surface the two rectangular banks of cells making up the memory array can clearly be seen, as can the fine wires connecting the chip terminals to the IC pins. Contamination on the chip should be blown clear. Any brushing is likely to disconnect one or more of the fine wires.

Having exposed the entire chip surface, the next job is to cover part of it up again. Surrounding the memory matrix are all the address latches, address decoding and buffer gates and if light falls upon them they will fail to control the memory matrix properly! A good lens system would ensure that no stray light endangered the control part of the chip. We tried paint as a covering agent but that was difficult to apply accurately, so in the end the sensitive chip areas were masked off using part of the metal plate. Soldering it back onto the IC resulted in undesirable flux splashes so finally it was glued back.

A small convex lens was positioned in front of the IC so that the chip surface lay in the focal plane of the lens. **Figure 2** shows how to find the focal length of a convex lens. In view of the fact that pieces of cardboard and plastic tape held the "lens„ in front of the "camera", it was perhaps surprising that any images were obtained at all. It will be very interesting to see what standard of results a school science department can achieve.

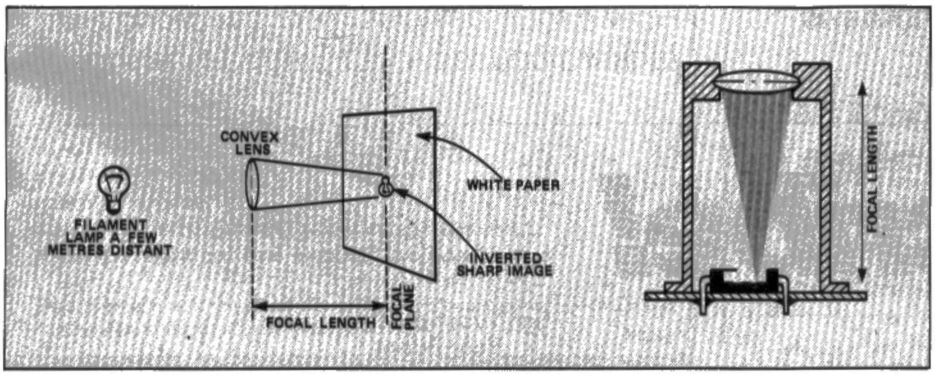


Figure 2a (left) shows the method used to determine the focal length of a convex lens. A piece of white paper is moved back and forth until an inverted image of a filament lamp some distance away is brought into sharp focus. The distance between the card and the lens is the focal length. Figure 2b (right) shows the lens mounted such that an image is formed on only one half of the DRAM's capacitor array.

CONTROLLING THE CAMERA

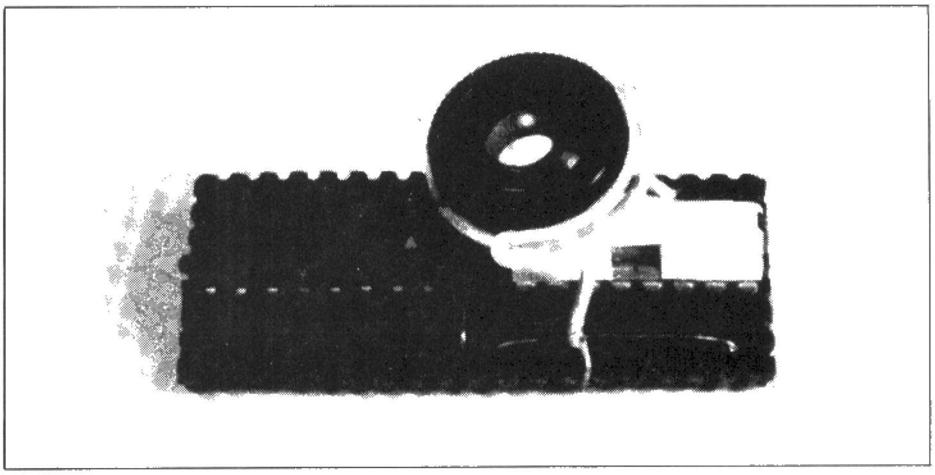
The logical arrangement of the memory matrix of any RAM chip can usually be gleaned from its specification sheet. In the case of the 4116 the 16384 memory cells are configurated as a 128 x 128 array which implies a nice square area of chip surface to receive the light rays. Unfortunately the

▶

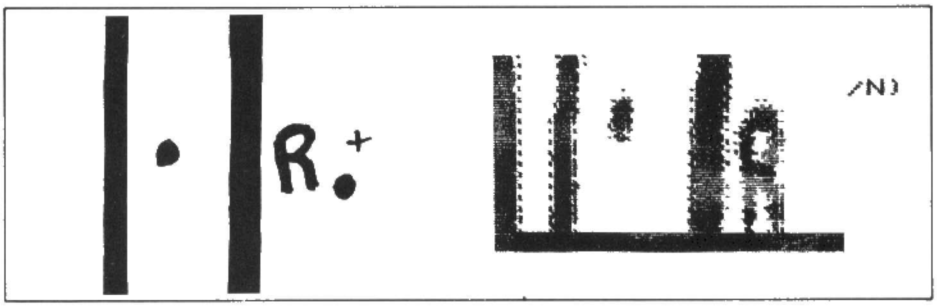
physical arrangement of the matrix might not be so convenient since the manufacturers never intended it to be used as a solid state camera! The 4116 is physically separated into two rectangles of 64 x 128 creating a sizeable "blind spot" in the corridor between the two rectangles. In fact the solution is to focus on just one rectangle, and the program thus processes 64 x 128 = 8192 image points, and puts them out on the screen as Spectrum pixels in ink or paper colour. This seemed reasonable for a prototype. Later versions using a better lens and 4164 memories could attempt to process a greater number of points.

PATTERN RECOGNITION

Having built a prototype camera which will at least distinguish between light and dark and "see" simple shapes such as circles, triangles and squares, you will now wish to progress to the stage of attempting to teach your computer to recognise simple shapes. In a series of recent articles by Mike James (February, March and May 1983 *E&CM*) the topic of Pattern Recognition was given a thorough airing, and sample programs to run on a Spectrum were given to illustrate the points made. The programs work without the need for hardware devices since they are illustration theory rather than proving practical applications. Mike James supposed that a low-resolution light-sensitive device might be built for tens of pounds . . . Now you have one for under a fiver!



Even fairly crude lens mounting arrangements can give acceptable results yet care taken in this area will give the best results from the system.



The author's test card (above left) was kept fairly simple. A typical image produced by the system is shown above right.

Next Month . . .
Software listings for the camera.

NEXT MONTH
IN YOUR ROBOT

WE OGLE THE OGRE

*The Ogre, a new low-cost robot arm from Staines of Essex, is claimed by its manufacturer to be able to lift up to 2lb weights. Pound for pound this is the best performance on the market today – **Your Robot** scrutinises the boast and gives Ogre a thorough benchtest.*

A DAY OUT AT COLNE

*Last month **Your Robot** took a trip down the Thames to visit Colne Robotics' factory at Twickenham. Our report reveals that this major educational robotics manufacturer has a few surprises up it's sleeve.*

PLUS

How a four foot high talking walking Skol tankard causes robofear and loathing.

Subscribe!

SUBSCRIPTIONS

We are pleased to announce that subscriptions to **Your Robot** are now available.

To receive a copy of **Your Robot** every month for a full year will cost only £3.00.

Send your order to:
Your Robot
155 Farringdon Road
London
EC1R 3AD
making Cheques, POs payable to **Your Robot**.

THE 'ROBOT PROGRAMME' PREVIEWED

Peter Mathews previews the latest production from the BBC's Computer Literacy Project.

The team that produced the 'Making the most of the Micro' series have recently been hard at work putting the finishing touches to a series of programs to be transmitted during March and April under the banner of 'computers in control'.

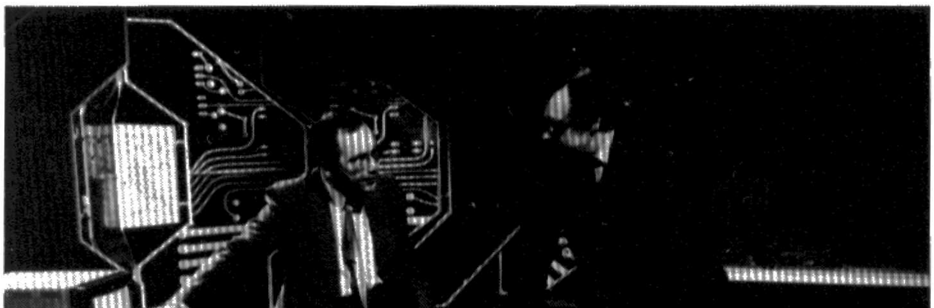
Ian McNaught Davies is again the front man of the series while behind the scenes David Allen (editor of the Computer Literacy Project) and Robin Mudge are responsible for production and direction respectively.

The series is concerned with the key areas of robotics and computer control systems and is divided into five programmes as shown in the programme schedule.

The support for this series of programmes is going to be as good or even better than "Making the most of the Micro". There will be a pack of programme notes and papers that will be available from the BBC Broadcasting Support Services Department and the National Extension College is planning a Control Course for owners of the BBC Micro. Demonstration software which will be seen and demonstrated in the series will be made available (see programme schedule) through the BBC's Telesoftware Service on Ceefax.

This will be much more ambitious in scope than the programs used to support Making the most of the Micro. It has been designed by the BBCs software development staff in a modular form. There will be aspects of the computer program that lights up lights. Others that drive motors and a whole range of control actjvities for hardware described and specified in the broadcast notes mentioned already. The Telesoftware computer programs, because they are modular, will enable the viewer to separate, for instance a motor control program from a lighting or sensor activating program. This will enable him to take those parts of the software that appeal to him out of the suite of programs and experiment with a specific section by itself. He could even take certain blocks out of the program and put them together to meet his own robot control needs. The ingenuity and flexibility of this approach will obviously create a considerable new dimension in experimental and educational aspects of the series.

If that is the software, then how about the hardware content? The BBC Buggy (already reviewed in the first issue of **Your Robot**) is a centre piece and is introduced in the first programme "Introduction to the Robot" along with Hero the educational robot manufactured by Heathkit Limited, a robot chess player, a machine for playing noughts and crosses and several other devices old and new. It is however the Buggy that keeps the pride of place as a ready made experimental platform for the viewer who participates in the programme and the Telesoftware that supports it. The first programme of the series also demonstrates computer control in action, for instance in a Lotus racing car where it modifies the driving capability of the machine by the modification of the suspension. A complex arrangement of accelerometers and analogue and digital converters does dramatic things to the handling and stability of the car. At the other end of the scale a demonstration of a Fisher Technic construction toy simulating a factory with all its controls at work at the Chicago Robot Exhibition is very eye catching.


John Coll and Ian McNaught Davis (courtesy of the BBC).

The programme "Making Sense of the Real World" in the series is all about sensors. The way that they can be linked to the micro computer so that it can monitor and react to its environment is a fundamental part of control. The experiments with light sensors and the associated experimental software show the fundamentals of control. It gives a basis of expanding these principles to the control of a hoist and on a different level the fast sampling of signals in weather mapping technology. Then just to show that lighting is not the only media for sensing, a smoke detector (with software), an electronic sensor for use in detecting alcohol on the breath (without software, courtesy of the Police) are all demonstrated. The most dramatic of these sensor controls however is in the wind tunnel developed at Southampton University. A model aircraft with a metal core is held in the centre of the tunnel by powerful magnets aligned all round the outside of the tunnel wall. The way that the model can be suspended and manipulated in space by remote control of magnetic force is an eerie thing to see.

The "Making Things Move" programme demonstrates many practical applications. It shows the control of external devices such as motors, solenoids and relays. Open and closed feedback techniques are examined as they are essential to the accurate control of these devices and they interface to the BBC micro with input and output capability. Practical demonstrations of how to develop and build control devices using a development system which is based on a Z80 or a 6809 chip give a clear outline of how to set about a project. The product which the programme demonstrates as such a project is the sensor for measuring the power of a turbine.

The six legged robot "Odetics" is shown probably for the first time on television in this country and very impressive it is. The legs each have a 16 bit processor on them and they are all talking to each other as the robot walks and moves its legs. It is a very clever control system.

The programme is concerned with problem solving and looks at microprocessors in dedicated control systems. A home control switching technique using the internal wiring of a house is looked at and also the Voltan voice input system as a demonstration; controls show a home security project which has very practical applications. The final programme examines the future for robotic technology. The installation in Carolina in the USA is probably the most sophisticated robot installation in the world. Use of speech and visual pattern recognition is shown and its importance to robotics' future technology. The series then finishes up by putting the whole lot into its industrial context by showing the control techniques in a complete FMS system.

This series is probably the most important single initiative that has been taken in educating the public in the use and possibilities of robotics and control systems in computers. The ability of all ages of viewer to, not only see but experience and experiment with, the control of hardware/software systems will give a real impetus in understanding the possibilities and techniques of the use of microcomputers in sophisticated mechanical control. Particularly when there is such an effort to back up the programme with notes and papers referring to both hardware design and Telesoftware programs in which to experiment. It shows the ability of television to inform and educate in the "black arts" of computing and high technology. ■

PROGRAMME SCHEDULE

The programmes will be concerned with the following aspects of control and robotics:

1. **Introduction to the Robot**
2. **Making Sense of the Real World**
3. **Making Things Move**
4. **Problem Solving**
5. **The Future**

The final titles of the programmes will be released in the Radio Times.

Friday	March	2	Programme 1 BBC 2 12.30
Thursday	March	8	Repeat Programme 1 BBC 1 23.30
Friday	March	9	Programme 2 BBC 2 12.30
Thursday	March	15	Repeat Programme 2 BBC 1 23.30
Friday	March	16	Programme 3 BBC 2 12.30
Sunday	March	18	Repeat Programme 1 BBC 1 12.35
Thursday	March	22	Repeat Programme 3 BBC 1 23.30
Friday	March	23	Programme 4 BBC 2 12.30
Sunday	March	25	Repeat Programme 2 BBC 1 12.35
Thursday	March	29	Repeat Programme 4 BBC 23.30
Friday	March	30	Programme 5 BBC 2 12.30
Sunday	April	1	Repeat Programme 3 BBC 1 12.35
Thursday	April	5	Repeat Programme 5 BBC 1 23.30
Sunday	April	8	Repeat Programme 4 BBC 1 12.35
Sunday	April	15	Repeat Programme 5 BBC 1 12.35

The programmes will have an accompanying software content. This will be available to viewers on the Telesoftware Service on Ceefax and is planned to be available for two weeks after the programme goes out ie The Software on the first programme entitled "Introduction to the Robot" will be available for two weeks after the first transmission on 2 March.

HERE'S LOOKING AT YOU

Peter Matthews casts his eye over robot vision systems and focuses on the Cyclops low cost imager.

The robot has been able "to see" since the 1960's, by using a TV camera and a large chunk of computer power. High memory costs have restricted these vision systems to simple tasks such as reading special characters or bar codes. But now a lowering in the cost of electronic technology has made vision dramatically cheaper. As a result the market is beginning to grow, the price may come down but unit sales are rising dramatically. The number of companies interested in this area is also rising. In America the vision market was only just over $20 million in 1983 but it is expected to reach $180 million in 1987 and almost a billion dollars by the middle 1990's. In Britain the market is about a fifth of the American and sales will grow probably at a rather lesser rate over the next decade, and most of these will come from customised systems. Only when the technology is better understood by industry, academics and the public will 'off the shelf' units come into their own.

Vision techniques fall into three broad classes. These are:-

Binary Vision which consists of image analysis by recognising just two codes, black and white or '1' or '0' to the computer.

Grey Scale Vision which ranges from black at one end of the scale through a series of contrast steps to white at the other. It also embraces a variety of techniques such as edge detection or matching greyscale patterns statistically.

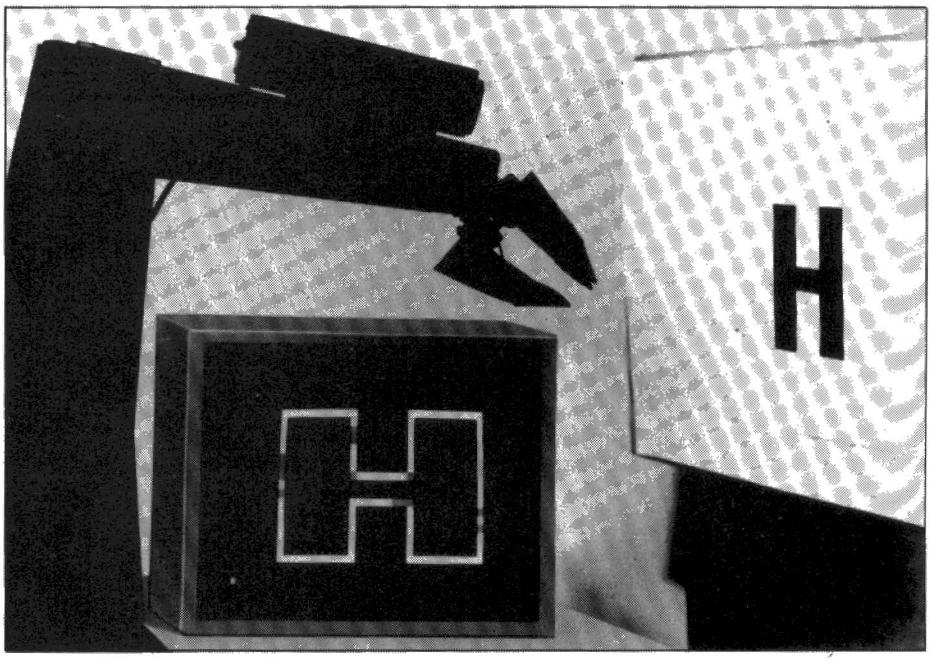
Light Striping is a sophisticated technique which consists of analysing binary images or workpieces on which stripes of light have been projected by a lamp or even a laser. The light stripes enable the vision system computer to estimate the distance to objects by triangulation and infer their shape and size from distortions in the light stripes. This is a clever technique but is not yet popular with the industrial user who prefers his system to be fast, simple and reliable rather than sophisticated. There are however an increasing number of interesting applications. Three-dimensional sensing of shape size, orientation and depth of workpieces enables a 'sighted' robot to pick out a single component from a pile of them even if the item is half buried in a pile of others.

ROBOT WORKERS

The inspection and checking of parts and products is something that gives specific employment to about one tenth of the total factory workforce in Britain. It is estimated that about a third to a half of all these workers will be replaced by vision systems. Yet an even more important role is robot guidance and control. Making a robot see and then make decisions on what it sees will make it very smart indeed. It allows a robot to handle unexpected or uncertain situations. For example, one of the most intelligent robot installations is concerned with manufacturing turbine blades using total automation. The robot selects a billet of steel, inserts it into a furnace and when heated to the right temperature is placed into a swaging machine for shaping. Before passing the finished blades the whole forging is automatically inspected with another vision system to check the machine tolerance. The "sighted" robot therefore has the ability to make judgements and act on them. It can even judge the temperature of the metal billet by the colour of the metal. Such techniques in forging and machinery make for skilfull intelligence that are human-like in character. This machine intelligence whereby vision is used and acted upon is a considerable step towards the unmanned factory.

The cameras, computer and software for this may cost up to £20,000 a system. Industrialists think that vision should be cheap, no more than 10% of the cost of the robot. There are those still working on a low cost vision system because they feel that the present cost level requires an act`of faith to purchase one. This is particularly true as the average production engineer cannot light or tailor the software of a serious vision device. This is the reason for a pilot scheme and base for experimentation for the new user. The Cyclops Vision System costing under £100 gives not only a digital camera but an experimental device with explanatory manual containing specific experiments in the applications using the basic techniques already mentioned. The applications are Scanning, Template Marking, Twisted and Grey Scales, Image Processing, Noise Filtering and Array Scanning. These are all used in experimental and practical modes and the PUMA robot is pictured using Cyclops in an industrial task. It is also being used to position a small robot in a laboratory workstation and has also been useful at Harwell, The Open University and The Laboratory of the Government Chemist.

The Cyclops manual explains the eye mechanism and the function of the rods as grey scale light intensity detectors. It then uses the analogy of the eye to demonstrate the similarities and differences to the vision system. The package consists of a camera, software and the already mentioned manual. The software not only drives the vision device but shows what is happening in diagramatic form on the screen. The explanation of the binary technique for example shows how the computer recognises a scanning



The robot is looking at the 'H' image as outlined in the article. There are some grey areas in the white outline of the H image. This is because there is some shadow 'noise' that the system has not decided is black or white. The program can be modified to go over areas of doubt again.

technique which then looks at an image which happens to be an H. The first step in acquiring an image is to form a 'map of pixels'. That is then recognised by both the computer and operator.

The next stage is called 'template marking'. A template maps pixels to bits and the 8-bit lines to bytes and therefore takes 8 bytes. The cassette program contains templates for all the letters of the alphabet which at 26 x 8 bytes takes 208 bytes of memory. The experimenter is encouraged to create his own templates such as workpiece outlines which can be used for practical purposes. These templates are then used in a 'marking' mode. The camera is shown an unidentified shape, such as the letter H and the system is then asked to identify it. The computer then indexes through the established memory locations in the templates. It sorts the numbers of 0's and 1's for all the letter or image templates. It keeps a score of the matching patterns after scanning each letter and then displays the best scorehits in comparing the image with the template which (we assume) is the H. By this method you and Cyclops have taught your computer to read the alphabet.

The computer can also be taught to recognise colours, or rather, grey scales. A newspaper photograph is made up of grey scale dots each of them corresponding to a colour somewhere between black and white. By shading areas with colour levels a recognisable image can be made. In theory the Cyclops is capable of recognising 256 grey scales but this is limited by the illumination of the image. For practical purposes it works with about 40 (a photocopier uses 32 steps of grey to produce its photographic image). The scales of grey are created by a light intensity which becomes a ramped function. The computer then changes into digits (0s and 1s again) and then holds that information in its memory. The Cyclops can be used with any computer with any user or games port and the software was developed for the BBC. It is being developed for all the popular computers.

"... Techniques fall into three classes ..."

Once this technique is acquired the important process of edge detection must be examined. This technique can locate holes (important for workpiece orientation), measure parameters of a shape – this can give the centre or centoid of the object, yields information about concave or convex images and many other shorthand methods of judging the properties of an object and identifying it. The system is not interested in areas of black or white so they are ignored, only the change from one colour level to another is of interest to the camera. An interesting part of manual/software presentation is the fact that the system is compared to biological systems and their feedback.

Reactions can be simulated by the vision system triggering a reaction from an attached robot or another mechanical device. The robot then can be fitted with at least vision similar to that used by primates and all at a very low cost.

It is obvious that good vision is essential for image systems of all kinds. Noise in the analogue/digital signal is a major problem and the most common static noise is shadows which makes it necessary to illuminate objects well. Another more difficult noise source is oscillation at or near to the threshold settings of the DC levels. A typical culprit is the fluorescent tube which when used as a light source fluctuates considerably.

A number of hardware and software experiments are suggested for noise elimination or at least limitation in the Cyclops manual. A software technique used in this area is that of 'autocorrolation' which uses several inputs of the image over several time or spatially shifted samples.

Higher ranges of vision technique, together with those described above will be on view at the Machine Intelligence Show and Conference on Robot Vision and Sensory Controls at the Cunard Hotel in London from 9th to 11th October 1984. ■

CROY-1

A report from Richard Moyle on a free roaming robot being developed by the Croydon Robotics Group.

In July 1983, several members of the Croydon Computer Club decided to form a Robotics special interest group. The aim was to design and build robots capable of development as technology and ability advanced. Some members of the group had experience in building small machines and mice, but wanted to go further than this, and build a self-contained machine of substantial size, and with a degree of autonomy – Croy 1.

Problems were found immediately. Large machines are heavy. They need big motors, using lots of power. This means large batteries – more weight! These large itemsd are also expensive. Big robots find it difficult to manouvre around furniture, and cause damage if they don't.

Eventually, the modular approach was adopted. Each part of the robot would be built as a separate project, and communication between parts would be by an internal command language, probably based on ASCII strings similar to Logo. This approach makes things much easier. In designing any unit we can concentrate on solving the problems associated with that unit, provided we ensure a standard interface is a feature of associated units. After all, the body unit is not concerned how an arm opens a door, only that it has done so, or that it cannot be done and why.

The first unit being constructed is the mobile platform. The base measure about 20" by 18", a size we feel is the largest suitable for domestic rooms. We have chosen to use a tricycle arrangement, with the single front wheel driven, and used for steering. This makes the control programming easier, though the initial mechanical construction is more difficult. The front wheel is mounted directly on a modified car windscreen-wiper motor. For steering, the whole assembly is turned by means of a chain drive from a second motor attached to the base. Both motors are controlled by fairly simple electronics, whose main function is to prevent damage from stalling or software faults. The rear of the platform contains a 12V car battery and charging equipment.

When complete, the base unit will contain collision sensors, and be capable of movement under the control of its own processor. As other units are added, the processor in each unit will communicate with the others under the supervision of a master controller. At present we expect this to be an on-board Spectrum running Microdrives and forth. For now, the motors are interfaced directly to a remote Spectrum pretending to be the rest of the system. Early tests on the far from complete base unit are very encouraging.

At some stage we may decide to change the base unit for another design. Tracks, legs or different wheel arrangements may prove better. Our approach means we can do this with the minimum of disturbance to the rest of the system, and no programming changes to the master controller.

While the field of Artificial Intelligence is a subject which will occupy a great number of people for a long time, there are still many areas of research purely in the mechanics of robot construction. For example, should the control system be based on a bus structure, so that all units can communicate directly with all others? If so, should it be parallel or serial? Perhaps a heirarchical system would be better, so the arm can only control body rotation with the intervention of the master. These questions, and many more, may be answered in the next instalment of Croy 1.

In the meantime, Croy 1 will be making a guest appearance at the ALCC Easter Spring Fair, at Central Hall, Westminster, together with other machines built by the group. The Robot Arena (first seen at Breadboard) will be available for the benefit of anyone wishing to display their machines, or as a meeting point to talk to other enthusiasts.

LANGUAGE IN CONTROL

High-level control languages impart a degree of intelligence to the robot. Gary Herman surveys those currently available.

Anybody with a couple of hundred pounds can buy a turtle or robot arm – ready-made or in kit form. For £1500 or so, you can get a fully operational free-standing robot with its own arm, gripper, sensors and means of locomotion. Having bought your device though, how do you get it to operate? In some cases, the device will come with a more-or-less primitive operating system using on-board microprocessors and ROM-based software to decode instructions input through manual controllers (known as teaching pendants), on-board keypads or interfaced host computers. More devices, however, come 'naked' – unadorned hardware with, perhaps, a minimal amount of logic circuitry. Even when your device *has* an operating system, it will commonly be restricted to providing a small set of programming and system commands: MOVE, SLOW, ELBOW, DELETE, DO and suchlike. There is hardly any applications software available to the hobbyist.

HARD-WIRING

A robot device is, of course, just a useless hunk of metal and plastic without a means of controlling it. Simple devices – for example, the Tomy 'Armatron' – are operated by hard-wired switches or analogue controllers, but such devices are hardly robots. One level up are elementary programmable devices like Milton Bradley's 'Big Trak' toy. This is a tank-like vehicle with an integral keypad. It may be programmed by pressing suitable keys – forward, reverse, turn so many degrees in such a direction and so on. Motion commands are entered using arrow keys and numbers representing linear or angular distances. There are also a number of keys marked with system commands – RPT (repeat), CK (check), CLR (clear), TEST and, of course, GO. Meaningful keypad entries, therefore, constitute a simple robot control language decoded by a Texas Instruments TM 1000 microprocessor (which is only available to OEMs). In effect, Big Trak has a hard-wired operating system which compiles programs (up to 16 steps long) written in the Big Trak control language.

Now, it is possible to interface Big Trak to a host computer by connecting the parallel output of the computer through a multiplexer (the CD4051 chip has been used for this purpose) to the appropriate pins of the vehicle's TM 1000 processor. It's then a simple matter to write a BASIC program to POKE or OUT suitable values to parallel output in order to control Big Trak. It would be quite possible to program more than 16 steps – ideally by specifying commands in 16-step blocks and using a closed loop to feedback end-of-block data to the host computer. A simpler system (using open loop control) would not require feedback and would just wait for a suitable period between transmitting blocks of instructions to Big Trak.

In such a system, the robot control language is implemented in BASIC. Using a multiple argument INPUT command, we could actually write a program to control Big Trak in what is, in effect, a higher-level language than BASIC itself. Or, using keyboard scanning commands like INKEY or GET, robot control could be implemented through BASIC using something resembling direct mode.

This example demonstrates all the features of robot control languages. In the case of Big Trak though, there is a considerable amount of hardware between the control program and the actual operation of the device. In designing a flexible robot system from the bottom-up it is preferable to avoid excessive hardware. Usually, this involves driving robot motors direct from a parallel port and for simplicity, this implies the use of stepper motors which do not demand closed loop control to operate with a degree of accuracy. (Powertran's Micrograsp arm uses servos but with only local closed loop control. The error signals are not, therefore, fed back to the host computer). Inevitably, the most sophisticated robots will utilise on-board processors, memory and software. This enables them to stand alone, to operate sensors and to 'tune' their behaviour as quickly as possible. However, the principles of robot control languages remain the same and – for our purposes – we shall assume that all control issues from a separate computer interfaced via a parallel port.

TEACHING PENDANTS

The earliest commercial robot programming systems – devised for Unimation's 'Unimate' robot arms – were not languages in the strictest sense of the word. Direct input was used to move the robot arm to a particular position whose co-ordinates could be stored in the system. By moving a joint at a time and pressing a 'record' button, the desired sequence of positions in space that the arm needed to pass through from start to finish was stored. At the end of the sequence of movements, the robot was instructed to perform a particular task (release a paint spray, for example) and this instruction too was stored. Keying in the command 'automatic' then resulted in the stored sequence of positions being taken up by the arm and the stored action being performed over and over again.

Such programming is still used today – usually by means of a 'teaching pendant' which takes the robot through a sequence of actions. Even a device as simple as Big Trak could, in theory, be programmed this way because it utilises photoelectric sensors to judge distances and angles moved. With the appropriate hardware modifications, it would be an easy matter to 'walk' Big Trak through its required sequence of actions and store the distance and angle data in RAM. Clearly, the benefit of this approach is that it demands no special programming skills on the part of the user. But the disadvantages are considerable – not the least being that the robot cannot change its behaviour in any way. It is not just difficult to instruct the robot in general terms (say, to find an object and move it) – it would be impossible.

A LITTLE INTELLIGENCE

The purpose of robot control languages is then not simply to enable a device to run through a series of specified movements automatically. If that was all we wanted robots to do, there would be no need for languages. The point of a robot control language – like that of any programming language – is to enable the robot to perform generalised tasks in which, in particular, conditional branches may be made.

Ideally a robot control language should be implemented in a high-level language which allows loops and conditional branches. The first such language was developed at Stanford University in order to control a hand-eye device being used to investigate the problems of artificial intelligence. The hand-eye was required to manipulate a simple blocks-world: finding blocks, sorting them and moving them. The Stanford system had four commands – Initialize, Servo, Open and Close. 'Initialize' set the hand-eye to a home position. 'Servo' specified joint positions, using numerical arguments. For a hand-eye using revolute geometry such a command might read Servo 20, 30, 43, 17, 180 – which would move the waist 20 degrees, shoulder 30 degrees, elbow 43 degrees, wrist 17 degrees and gripper 180 degrees, with respect to the home position or relative to the last position. 'Open' opened the hand-eye's gripper and 'Close' closed it. Such a system is obviously more complicated than the use of a teaching pendant, but by embedding the commands in a high-level language as subroutines it is possible to branch to instructions if particular conditions pertain. In other words, the robot can change its behaviour.

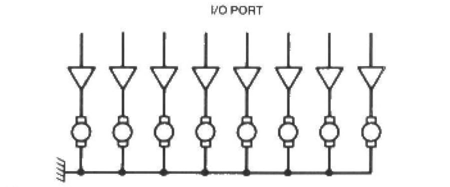


Figure 1. It's possible to drive steppers from individual data output lines.

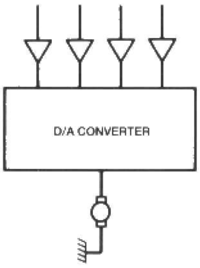


Figure 2. A more elaborate system might use binary coding to drive a single motor with variable speed, say, using a latch and D to A conversion.

Refinements of the Stanford system involved 'disassembling' elementary commands. That is, sophisticated control languages were developed in which commands like 'PICK UP BLOCK' were translated into sequences of absolute or relative movements and actions controlled by lower-level commands like Servo, Open and Close. The most successful of these – introduced in the 70s – were Algol-based languages: AL, PAL and VAL, in particular. PASCAL – a latter-day Algol derivative widely available for micros – is today a popular language for robot control.

USING BASIC

As we've seen, it's quite possible to use BASIC as the foundation of a robot control language. As long as your version allows POKEs or output commands (OUT, for example), and your computer has an accessible output port, it is a simple matter to write your own robot control language. It would be only marginally more complicated to write input routines to allow feedback from position, touch, light or auditory sensors. In the simplest implementations, each data line at your computer's input/output port can be directly connected to a single input or output device – for example, touch switches for input or stepper motors for output. Pulsed signals on the data lines then provide the bit patterns transmitted or received by the robot control program **(Figures 1 and 2).**

For many implementations, BASIC is considered too slow or too unwieldy – especially since sophisticated robot motion demands a great deal of number-crunching. However, a robot control language can be implemented in any language. The Oregon based Intelledex company has developed Robot Basic which is implemented in Microsoft BASIC, while American Robot has implemented robot control languages in BASIC, Fortran, C and Pascal all written for a Unix-like operating system.

Off-the-shelf hobbyist software is still thin on the ground. RB Robot of Colorado has developed Robot Control Language for its RB5X personal robot. The RB5X contains an on-board controller using Tiny BASIC – an integer-only subset of BASIC. RCL – so far written for Apple computers only – is an assembler which translates (according to the company's publicity) 'English words' into Tiny BASIC.

ROBOFORTH

A possibly more useful approach is offered by Cyber Robotics of Cambridge, whose Cyber 301 robot comes complete with RoboForth – a robot control language which is a superset of Forth. The first Cyber robots used BASIC to implement a control language based on Unimation's professional-standard VAL language. However, the decision to utilise Forth was taken when it was decided to design a robot arm controlled entirely by a host computer. For this purpose, BASIC was considered too slow and inefficient. Forth – originally written to control radio telescopes – is admirably suited to robot control applications. A typical command might be S+ 50 MOVE, which would move the 301's shoulder through 50 steps downwards. At first sight, this may seem no superior to a similar command written for a control language implemented in BASIC. It is in the nature of Forth, however, that such a command will be compiled more efficiently and executed more rapidly in that language. Other languages – especially the Lisp-based ones like Logo and Prolog which, like Forth, allow recursion – may also prove better suited to robot control than BASIC. Logo, in particular, has 'turtle commands' built into it (FORWARD, BACK, RIGHT and LEFT) which can already be used to control real, rather than merely graphic, moveable devices.

The scope for development is great, especially in the area of generalised control languages incorporating closed loop control, input techniques, search algorithms and non-specific commands like PICK UP, FIND and STACK. A flexible implementation language like Forth, I/O ports and a little imagination are all that's required. ■